A PARALLEL SLICING ALGORITHM FOR SOLID FREEFORM FABRICATION PROCESSES

C. F. Kirschman, C. C. Jara–Almonte
Center for Advanced Manufacturing
Mechanical Engineering Department
Clemson University, Clemson, SC

ABSTRACT

Slicing can account for more than 60% of the time to prepare a part for building on a stereolithographic apparatus. To improve the preparation time, a parallel slicing algorithm was developed. The algorithm was run on a Butterfly GP1000 using 2, 4, 8, 16, and 32 processors and superlinear speedup was observed due to high memory requirements. The parallel algorithm can reduce slice times by up to 92% on 16 concurrent processors as compared to a single processor.

INTRODUCTION

Stereolithography is one of several different Solid Freeform Fabrication (SFF) technologies available today (Ashley, 1991). The most common stereolithography apparatus (SLA) is the SLA 250 from 3D Systems (Valencia, CA). Because of its popularity, the initial software development described in this paper was aimed at this machine; however, it is believed that the concepts can be extended to any of the technologies.

The preparation of a part, detailed by 3D Systems (1989), begins in a CAD system. A solid model is created to represent the part. The part is then converted to a stereolithographic format (stl) file. This file contains a set of triangles which define the surface of the part and the outward pointing normals to these triangles.

Next, this faceted representation is preprocessed for building in the SLA. First the part is supported, and then it is sliced by another computer. Then the files containing the layers are sent to a third computer for merging, preparation, and reconstruction in the SLA. The parts are built layer–by–layer from the bottom up in layers 0.0025 to 0.030 inch thick.

One of the biggest complaints of the users of this technology is that the preparation time before building is too long. Slicing in particular can account for 60% or more of the time between the CAD system and the SLA. Because of this, efforts are underway to reduce the slicing time. This paper discusses techniques for improving the slicing speed by employing parallel architectures.

SLICING

The slicing process is depicted in Figure 1. A solid model is designed in CAD, and then is tessellated. Each of the resulting triangles is then segmented, and the segments for each z–level are joined to form a contour. The contours define the shape of the part to be built.

Currently, slicing at Clemson University is done by a simple geometric algorithm (Chalasani, 1991). The algorithm is shown in Figure 2. First, the stl file is read in. Next, the coordinates of the vertices of the triangles are multiplied by a large number, usually about 5000. This converts the

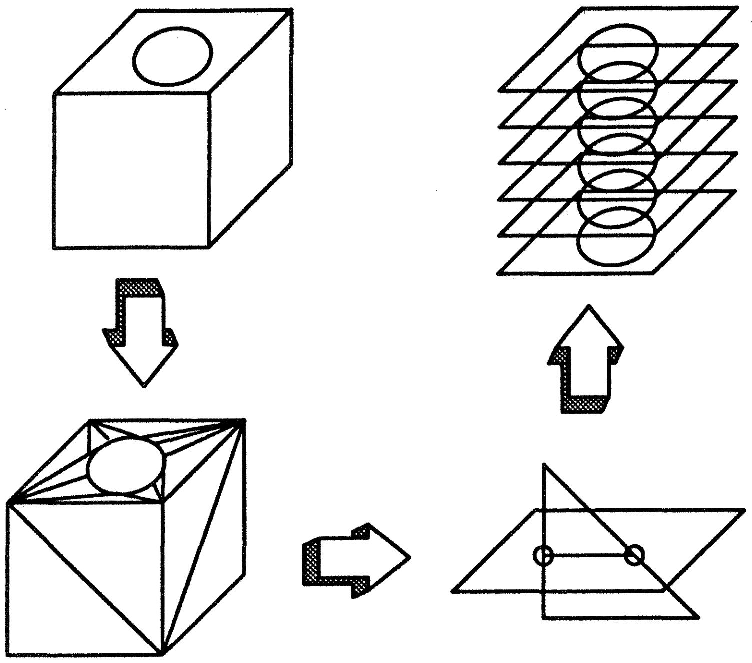

Figure 1. The Slicing Process

coordinates to integers, which computers can process faster. Then all flat triangles are separated and identified as the skin fills on the top and bottom of the part.

Then, for each z layer, each remaining triangle is checked to see if it passes through a plane parallel to the platform at that height, using equations such as those in Wolstenholme (1971). If it does, the intersections of the triangle and the plane make up a line. Once all of the triangles have been checked, the resulting list of lines is sorted in a head–to–tail fashion. This leads to a complete contour of the part at this level. The area defined by this contour is then hatched in a reasonable pattern, depending on the application; often squares or triangles are used. Hatching algorithms are discussed in Foley et al. (1990). These layers are then stacked up and written to a file, and the memory is freed. This sliced file is sent to another computer for the merging process. The merged file then is used to drive the SLA.

It is not necessary for the part to be completely solidified in the SLA, so the boundaries are drawn and hatched in a honeycomb manner and the part is built encapsulating liquid. The "green" part is then postcured in an ultraviolet (UV) oven until completely solid (3D Systems, 1989). This reduces wear on the laser, which has a finite life.

ARCHITECTURE

The slicing algorithm shown in Figure 2 was run on both serial and parallel computers and compared for slicing speed. Although the different computers do not employ the same processors, they represent typical machines available today. The parallel and serial algorithms were kept as similar as possible, but certain modifications had to be made for different platforms.

Serial Computer

The serial computer used for this work was a Sun Microsystems SparcStation 1+ running SunOS 4.1.1. It uses a Sun 4/65 processor running at 25 MHz with a 25MHz floating point processor, capa-

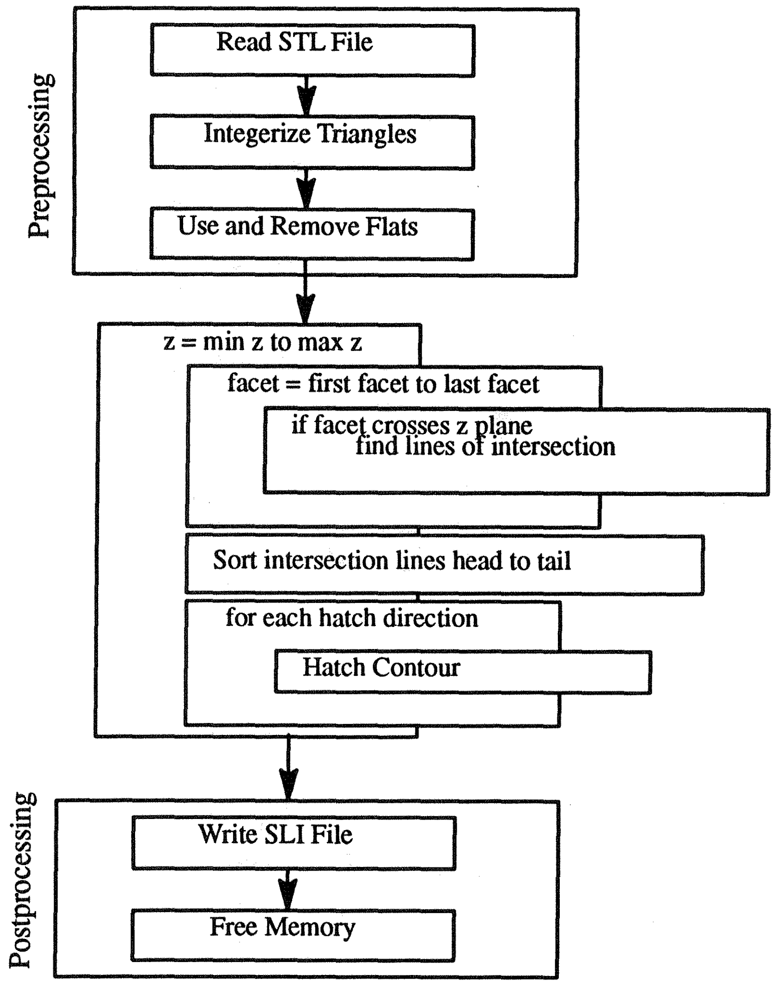

Figure 2. Geometric Slicing Algorithm.

ble of 15.7 MIPS. The particular machine has sixteen megabytes of memory and 100 megabytes of swap space. The machine was completely unloaded when testing was done.

Parallel Computer

The parallel computer used for this work was the Butterfly GP1000 running the MACH 1000 operating system, which is UNIX–like. It has 94 nodes, each one comprised of a Motorola 68020 processor with a floating point unit (FPU) and memory management unit (MMU). It is a physically distributed, logically shared memory machine as shown in Figure 3. The butterfly architecture is used because all of the processors are the same distance from the memory, and therefore the communication cost is constant. This provides better results when the algorithm is run on larger numbers of nodes. The actual program was run on clusters of 1, 2, 4, 8, 16, and 32 nodes of this machine.

TEST CASES

There were four test data sets for the algorithms, detailed in Table 1 below. There are two parameters which are important to the speed: number of triangles and number of layers. It was found that the layers were the controlling factor in problem size, so a formula was devised to represent problem size as a function of these two numbers. It was arbitrarily decided that the layers accounted for 75%

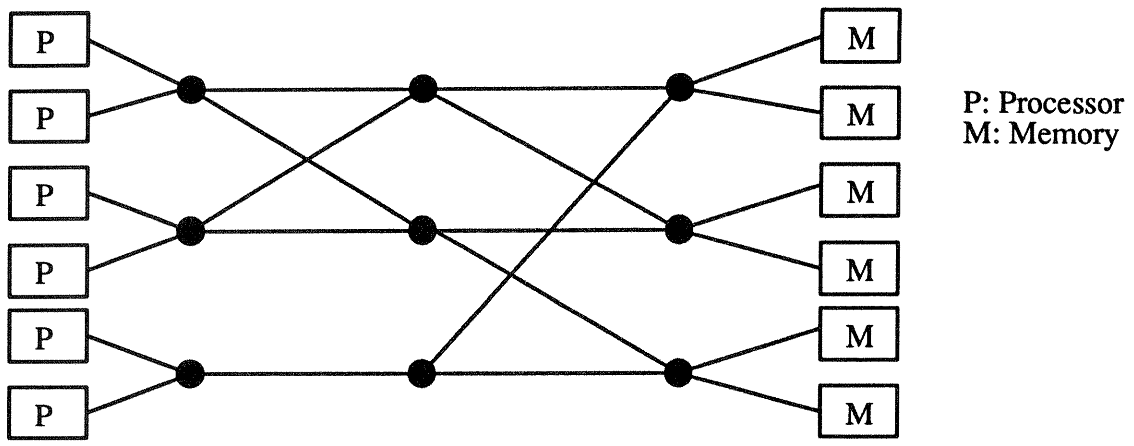

Figure 3. Simple Butterfly® Multistage Switch Connection.

of the running time and the number of triangles accounted for 25%. But, the number of triangles is about an order of magnitude greater than the number of layers, so this it was scaled by dividing by 10. A measure of problem size is given by:

$$\text{size} = 0.25\frac{N_{tri}}{10} + 0.75N_{layer} \tag{1}$$

where N_{tri} = Number of Triangles
N_{layer} = Number of Layers.

Table 1. Problem Sizes.

Problem Size	152	180	184	292
Number Triangles	40	2450	1008	9386
Number Layers	201	159	211	76
Slicing Time*(min)	5:23	6:05	9:21	13:48

*Sliced using 3D System's slicing algorithm on NEC 386/16, standard isohatch, no skin fills.

PARALLELIZATION OF THE ALGORITHM

The next step in the process was to parallelize the slicing algorithm. The Butterfly uses a "Uniform System" to support its parallel functions. It uses a "host" which sends jobs to the nodes in the cluster. This provides an easy method for the programmer to access the parallel power of the architecture. Essentially, the host serves as a manager, which passes out the tasks to each processor as it finishes its current task. This can lead to a bottleneck if many processors are waiting in the queue for jobs.

The essential differences between the serial and parallel programs are in the areas of variable management. Even though it is a shared–memory architecture, certain things must be declared global if the other nodes are to see them. The global declaration consists of a macro that informs all nodes in the cluster of the location of the memory, and allocation routines to gain space for these variables. Also, any data to be accessed through global pointers must be declared in the shared memory, which is also accessed through calls to the Uniform System.

Global variables are necessary due to the minimal nature of the calling routines for parallel node utilization. These allow the user to call a function with the only passed variable being the current index. Therefore, any other necessary information must be maintained globally. This creates a communication cost problem when too much information must be accessed from the global store or several processors need to write to memory on the same processor.

RESULTS

Serial Algorithm

The serial version of the algorithm was run on the SparcStation, and the computational times are in Table 2 below. The SparcStation uses a much newer and faster microprocessor, so the times are faster than the concurrent processors and the 3D Systems algorithm (Table 1). However, they show the possibilities for speed when better processors are used in parallel. The variations in these numbers are as great as 15%, with problem 184 varying by 5 seconds over 6 runs.

Table 2. Serial Algorithm Times.

Problem Size	152	180	184	292
Time (sec)	10	27	58	50

Parallel Algorithm

For parallel algorithms, it is useful to define certain measures of performance. From Fox et al. (1988), define T_{seq} to be the time elapsed on a single or sequential processor of the parallel machine. $T_{conc}(N)$ is then defined to be the elapsed time on N concurrent processors. Now, the concurrent speedup, S(N), can be defined by:

$$S(N) = \frac{T_{seq}}{T_{conc}(N)} \qquad (2)$$

If S(N) = N, it is called *linear speedup*, which means that the algorithm is twice as fast on two processors as one, and twice as fast on four as on two. Generally, linear speedup cannot be attained because of *communication costs*, which is the time it takes for one processor to communicate with other processors. Other factors affecting performance can be found in Fox et al. (1988). A second measure of the performance is the efficiency, ε, defined by:

$$\varepsilon = \frac{S(N)}{N} \qquad (3)$$

Efficiency is optimally 100%, but again other factors can influence this. Both efficiency and speedup depend on the performance on a single processor of the parallel machine.

This algorithm parallelized well. The times for each run are shown in Figure 4, and the speedup and efficiencies are shown in Figures 5 and 6. The speedup is actually superlinear and the efficiencies are greater than 100% for the 184 and 292 cases. This is due to the amount of memory these large cases require. For a single processor, the memory requirements exceed the locally available memory, so the CPU must access slower memory, either on another node or on disk as swap space. This makes the single processor case slower, which then skews the speedup so that it seems to be superlinear when in fact it is not. The small case, 152, shows this since it does not approach linearity or peak efficiency.

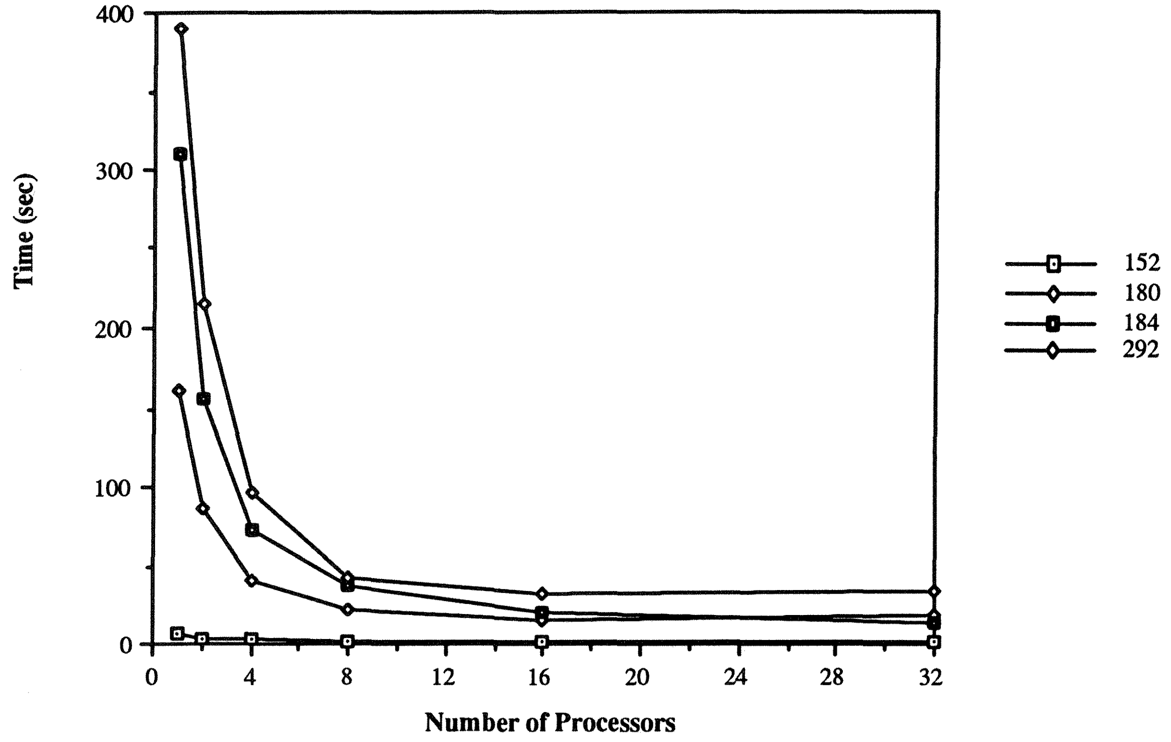

Figure 4. Time vs. Number of Processors for Parallel Algorithm.

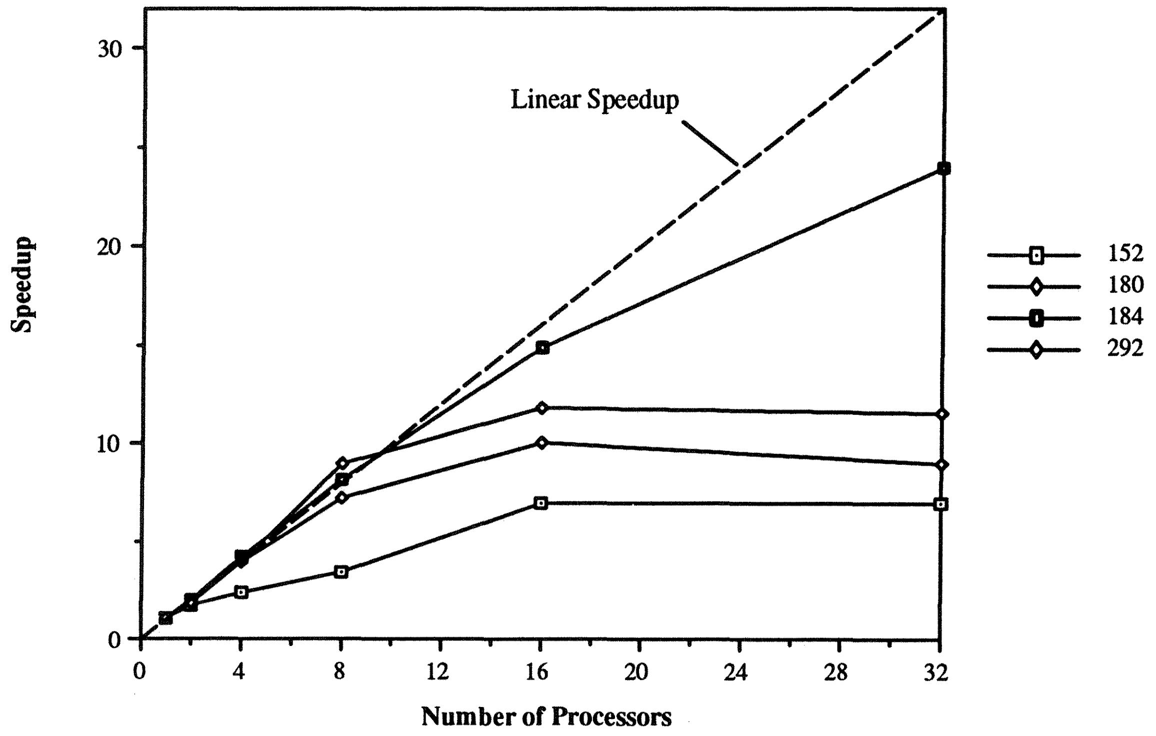

Figure 5. Speedup for Algorithm 1.

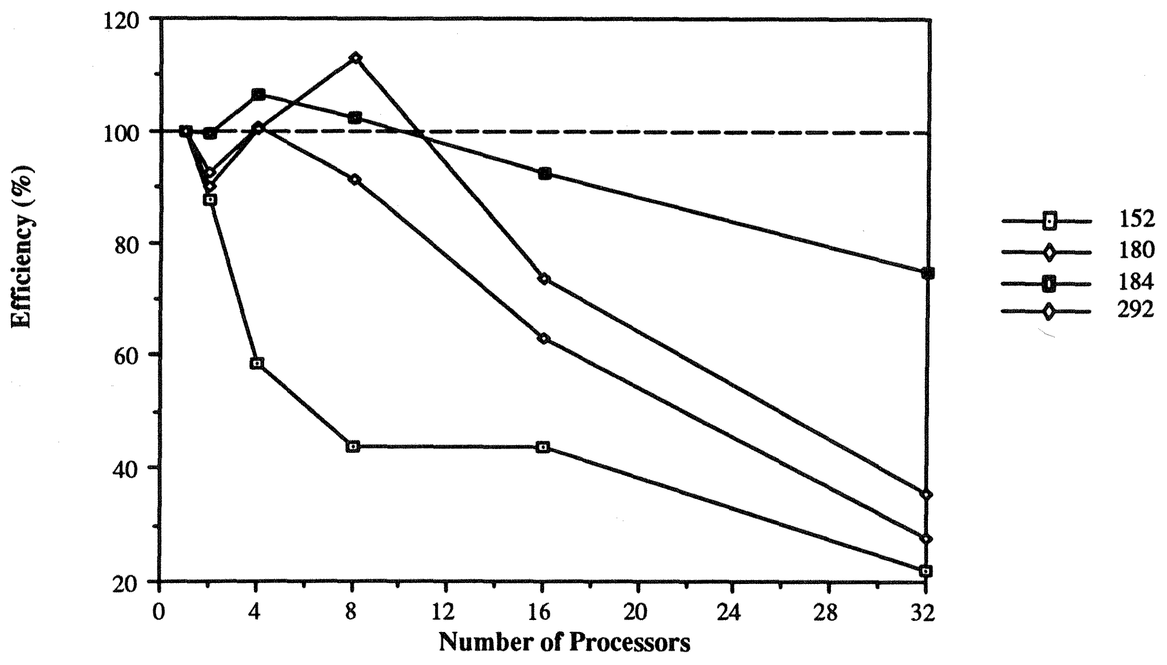

Figure 6. Efficiency for Algorithm 1.

The small case also suffers greatly from communication cost as the number of processors increase. Due to the small size, each processor has a minimal amount of work to do for each function call. This means that it will finish quicker and request another job from the Uniform System. However, with many nodes finishing quickly, the host will be unable to send them jobs sufficiently rapidly and a queue will form. This applies to larger problems when utilizing many processors. A solution would be to send more work to each node, perhaps several layers at once.

It is believed that the actual speedup is close to linear, but not actually superlinear. The efficiency is close to 100% with up to 8 processors, but drops off for 16 and 32 processors as communication cost increases.

CONCLUSIONS

The algorithm is a good implementation to exploit the power of the Butterfly. Superlinear speedups were recognized due to the large memory requirements. Even greater speed might be gained by a parallelizing smaller parts of the already parallel routines; this is known as *fine grain parallelization*. However, communication may become prohibitive if this is done. Other improvements could be gained by passing multiple layers to each processor when the number of processors becomes large. This will increase the work load on each processor, and reduce the queue at the host.

Furthermore, this algorithm can be implemented on a distributed memory machine. These machines, such as the NCUBE/10 hypercube, use a similar host to the Butterfly. However, the memory is not shared, so all necessary information must be sent to the node and returned to the hosts; this may increase communication costs. These machines are more common, and often employ faster processors than the Butterfly.

The time to slice a part on the Butterfly in serial is much greater than on the sequential SparcStation, due to the much slower processors employed by the parallel machine. However, when four or more processors are used, the parallel machine is faster than the serial machine. This shows promise for improving slicing speed by using faster concurrent processors.

The concurrent algorithm shows promise for reducing the preparation time for parts built on the SLA. With four processors, the slicing time was reduced as compared to the single, faster Sun processor. The improvements are small for small cases, but it is believed that the very large cases, which can take hours to slice, can benefit greatly from this algorithm. Furthermore, it is believed that the addition of skin fills to the algorithm will simply offset the curves by a fixed amount. All SFF technologies can benefit from increased slicing speed.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Xian–He Sun for supplying access to the parallel architecture and documentation used in this paper. Also, W. Gregg Stefancik, who helped develop the initial slicing algorithm upon which this work was based. Finally, we would like to thank Elaine T. Persall for providing relevant data on SLA operation.

REFERENCES

3D Systems, Inc. (Valencia, CA), 1989, SLA–250 Users Reference Manual, 1989.

S. Ashley, "Rapid Prototyping Systems", *Mechanical Engineering*, April 1991, pp. 34–43.

J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, Computer Graphics: Principles and Practice (2nd Ed.), Addison–Wesley 1990.

G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors: Volume 1, Prentice Hall, 1988.

K. L. Chalasani, B. N. Grogan, A. Bagchi, C. C. Jara–Almonte, A. A. Ogale, and R. L. Dooley, "An Algorithm to Slice 3D Shapes for Reconstruction in Prototyping Systems", *Proceedings of 1991 ASME Computers in Engineering Conference*, pp. 209–216, August 1991.

Wolstenholme, E. OE., Elementary Vectors, 2nd. edition, Pergamon Press, 1971.