Digital Data Processing Strategies for Large Area Maskless Photopolymerization

Anirudh Rudraraju, Suman Das, Georgia Institute of Technology

Abstract:

Large Area Maskless Photopolymerization (LAMP) utilizes scanning spatial light modulators that require layer slice data in the form of high-resolution bitmaps. Three different strategies have been implemented to fill this need. First, bitmaps were generated by direct slicing of CAD models using Spatial Technology's ACIS kernel. Second, bitmaps were generated from STL files through ray-tracing. Finally, an approach involving reconstruction of topological information from STL files for efficient slicing and image generation is being developed. This paper gives a brief description and implementation details of each of these strategies as well as data compression techniques being pursued by the authors. This work is sponsored by DARPA grant HR0011-08-1-0075.

Introduction:

Large Area Maskless Photopolymerization (LAMP) technique is an integral rapid manufacturing process which uses a spatial light modulator to expose whole areas in a single flash. The spatial light modulator in our case is a Digital Micro-mirror Device (DMD) chip developed by Texas Instruments. It is essentially a small chip with millions of tiny mirrors that can be turned on and off by feeding an 'exposure bitmap'. Light from the UV-source is shone onto the DMD chip and the various mirrors turn on and off accordingly to project the input image on to the photocurable resin surface. The head, consisting of the light source and the DMD chip, raster scans the entire exposure region which gives much higher speeds than conventional prototyping processes where typically a laser beam is used to raster scan the exposure region line by line. Also, because of the high scanning speeds extremely large areas (of the order 100 square inches) can be exposed without compromising on feature resolution. The feature resolution achievable in this process is only limited by the size of the tiny mirrors on the DMD chip (16-17 microns). Thus using this technique, macro sized parts with extremely high resolution features can be obtained with good process speeds.

The DMD chip takes high resolution bitmap images as input for exposure. Consequently all the CAD slice data needs to be output in this format. Most commercial slicing programs output slice data in the form of contours or scan patterns specific to the prototyping process they are designed for. Moreover, they work mostly only on STL files which are just an approximation of the original CAD models thus compromising on feature accuracy and resolution. Also, these commercially available programs do not offer much flexibility to modify the output data to suit the needs of the LAMP process. Hence new data processing algorithms that can efficiently handle both native CAD files and STL files for generating slice data specific to this technique need to be implemented.

Also, owing to the high feature resolutions (16-17 microns), slice images need to be generated at very high resolutions (1500 dpi). This can generate enormous amounts of data. Each uncompressed slice image is typically of the order of several megabytes, and at this scale the slice data for an entire part can run into several terabytes. Having data on this order of magnitude poses severe storage issues and also hinders fast data transfer between the controlling computer and the prototyping machine. Thus, efficient data compressions schemes and innovative data transfer techniques also need to be investigated.

In this paper, the various slicing strategies, both direct slicing and STL file slicing, investigated so far have been presented. The basic approach and the brief implementation details of each of these approaches are described in the sections that follow.

Direct Slicing:

In the Rapid prototyping industry 3D CAD models are first tessellated and converted into an intermediate file called the STL file format. This format has become the defacto industry standard for slice data generation. An STL file just contains a list of all the facets in the model in no particular order. Since it is a mere approximation of the original CAD part, many inaccuracies occur in the tessellated model and hence unsuitable for high resolution builds. Through LAMP, we intend to achieve feature sizes of the order of 16-17 microns and hence the more preferred approach would be to directly slice the original CAD models without the intermediate meshed approximations. Consequently, a direct slicing approach has been investigated. There has been some work in this area of direct slicing in the past. Guduri et al. proposed a direct slicing method for slicing a constructive solid geometry (CSG) representation of a part [1]. Vuyyuru et al. directly sliced solid models built by the SDRC’s I-DEAS and segmented NURBS-based (non-uniform rational B-spline) contour curves [2]. Chen et al. directly sliced PowerSHAPE Models [3]. Cao et al. proposed a method for directly slicing AutoCAD models [4]. In this paper, we present an approach for directly slicing CAD models using the ACIS kernel.

I) Using ACIS:

The ACIS kernel is a commercially available 'C++' CAD library marketed by Spatial systems, a subsidiary of Dassault Systemes. It offers robust APIs and function calls for most of the basic CAD operations. These APIs have been integrated into the slicing software to produce CAD slices. The produced CAD slices were then 'rasterized' to obtain the bitmaps used for exposure. The basic algorithm is outlined in Fig. 1.

The original CAD part that needs to be sliced is first loaded into the Algorithm using ACIS's load functions. ACIS libraries can only work with the "SAT" file format and hence CAD files in other formats need to be converted into the SAT format either by using commercial CAD softwares or by using ACIS's inbuilt file format translation functions. During the translation, numerical or topological inaccuracies could creep into to the part. In severe cases, error checking and correction schemes need to be implemented. Once the part has been loaded, its bounding box is computed to get an estimate of the size of the bitmaps that would be generated. A slicing plane is then created and intersected with the part using ACIS's Boolean APIs to get an intersection wire. Once the intersection wire is obtained, it is "rasterized" to obtain the bitmaps. This essentially involves shooting rays for each row of pixels in the image and computing the intersection points with the intersection wire. Pixel values are then filled with alternating white and black segments in between each of these intersection points as shown in Fig. 1 (e). The bitmap images obtained are then compressed using CCITT fax4 compression scheme which compresses the data by three orders of magnitude without any loss. (CCITT fax4 is an industry standard lossless compression scheme for efficiently compressing 1-bit TIFF images). These bitmaps are then fed to the DMD chip for exposure.

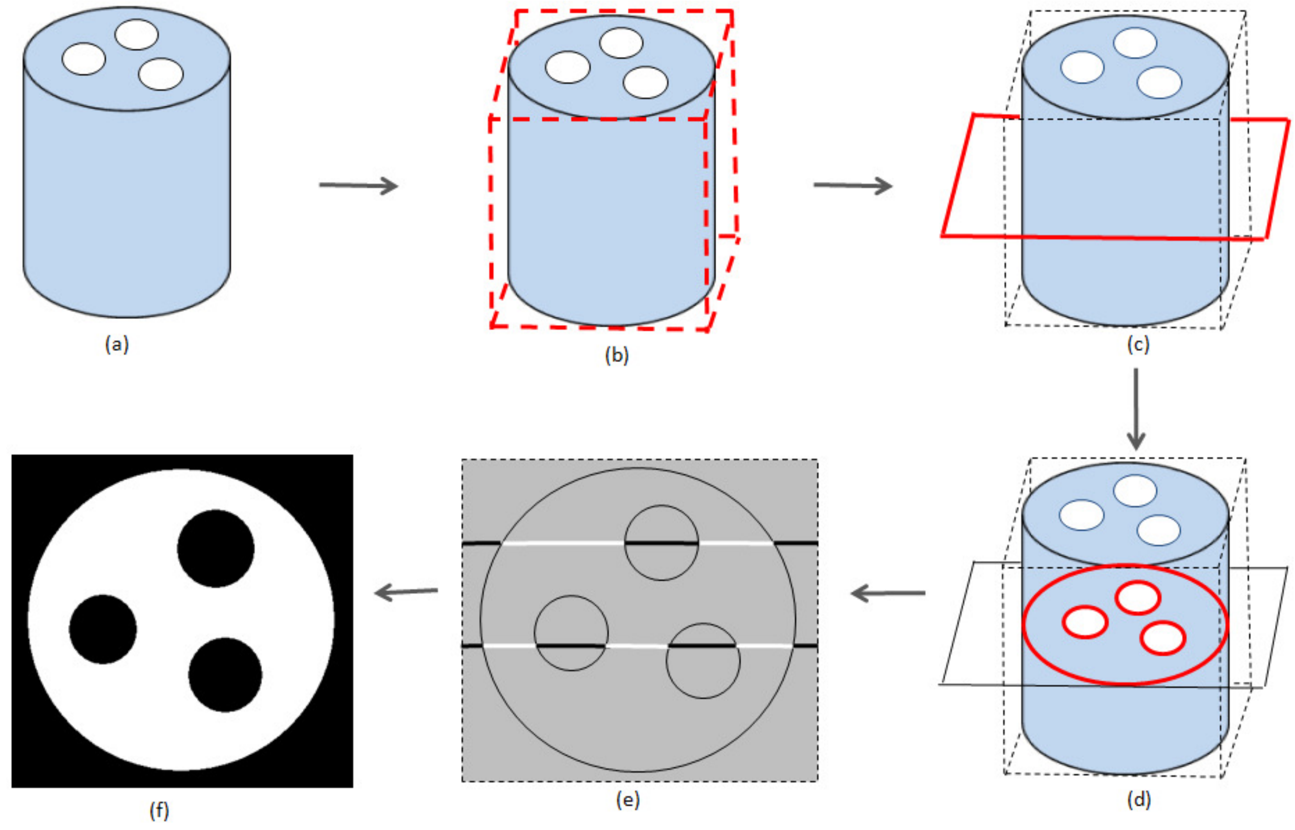


Fig. 1: Direct slicing algorithm: (a) Original CAD part loaded into the program. (b) Bounding box computed. (c) Slicing plane created. (d) Intersection between the part and the slicing plane computed. (e) Intersection wire rasterized to create bitmap images. (f) Slice bitmaps obtained and compressed in CCITT FAX4.

STL file slicing:

Since STL files are the defacto industry standard and the direct slicing approach using ACIS cannot handle these files, several alternate methods have been investigated. The implementation details of each of these approaches are given in the following sections.

II) Using POVRAY:

POVRAY is an open source ray tracing software that is routinely used in the computer graphics industry to render high resolution photo realistic images. The schematic in Fig. 2 shows the principle of ray tracing:

A small routine was used to convert an STL file into a POVRAY recognizable mesh object. The object is then sliced with a thin cuboid (thickness equal to slice thickness) using POVRAYs Boolean functions. The resulting intersection body is then positioned and ray traced as shown in the Fig. 2. The positions of the light sources and the camera are adjusted so as

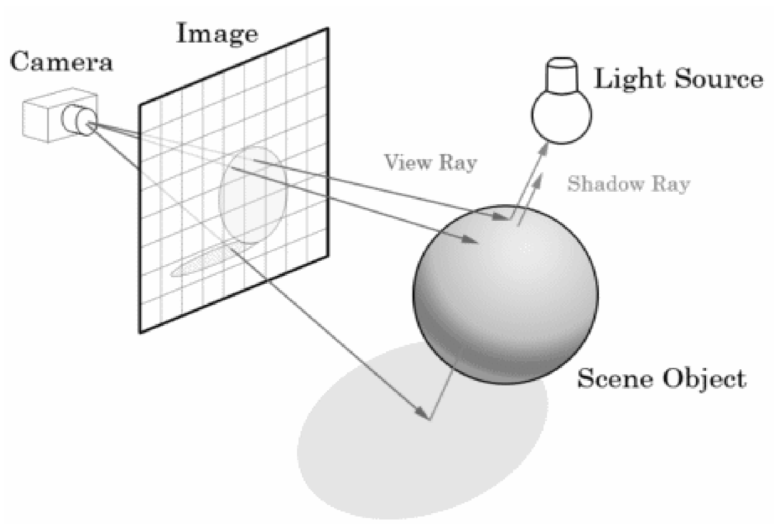


Fig. 2: Concept of ray tracing.

to obtain the slices at the correct scales. An "inside vector" was defined to identify the inside from the outside. When the scene is rendered, the camera shoots a ray for every pixel in the image to determine if it is inside the slice object or outside thereby rendering the slice bitmap image. The images obtained were then compressed using CCITT Fax4 scheme and sent to the DMD chip for exposure.

III) Reconstructing Topology of Triangular Mesh:

One of the major weaknesses of the STL file format is that it just contains a random list of facets in no particular order and does not contain topological information. If the topology information (e.g. edge connectivity, adjacency, no. of shells in the model) can be constructed using information in the file, it can be used to populate the ACIS data structure from bottom up and thus could be used to translate the STL file into a format that ACIS can deal with. ACIS's robust feature set can then be used to manipulate and slice the STL file. Also, constructing this topological data using information from the STL file will enable us to implement error correction and file manipulation routines which could prove to be very powerful in many applications. This was the premise in investing time in researching this method for slicing STL files.

There are a quiet a few data structures and algorithms available in literature to rebuild the topology information from a "bucket" of facets. Stephen Rock and Michael Wozny presented an algorithm for extracting the adjacency information and for implementing basic error detection/correction [5]. There are others in the computer graphics and visualization domain that address this issue [6] [7] [8]. Many of these algorithms use complex multidimensional data structures for sorting three dimensional data which are difficult to implement. J Rossignac et al.

of Georgia Tech proposed a simple integer array based data structure called the corner table data structure that is an elegant solution for working with triangular meshes [9, 10]. This data structure has been leveraged for our purpose and brief implementation details are discussed below.

Corner Table:

The Corner table data structure stores all of the topology and connectivity information in two simple integer arrays. The schematic in Fig. 3 shows the nomenclature and the integer arrays that hold the information. The region around a vertex in a facet is loosely referred to as a 'corner'. The vertex that corresponds to that corner is referred to as 'v(c)'. The corner opposite to the current corner 'c' is referred to as 'o(c)'. The left and the right corners are respectively referred to as 'l(c)' and 'r(c)'. The next and previous corners are given by 'n(c)' and 'p(c)' respectively (assuming the vertices are listed in a counterclockwise manner). The triangle to which the current corner belongs is referred to as 't(c)'. [Refer to figure 3(a)]. The two integer arrays that store the connectivity information are the Vertex array 'V[c]' and the Opposite array 'O[c]' as shown in figure 3(c).

For any given corner 'c', the corresponding vertex and opposite corner indices can be obtained from the Vertex array 'V[c]' and Opposite array 'O[c]' respectively. Once these two arrays are populated the adjacency information is available. For example, starting from a random corner 'c', we can access the left triangle by querying t(o(p(c))), the right triangle by t(o(n(c))) and the opposite triangle by t(o(c)). An edge array 'E[c]' can also be constructed in a similar manner to store the edge connectivity information. It would store the edge index of the edge opposite to a given corner 'c'.

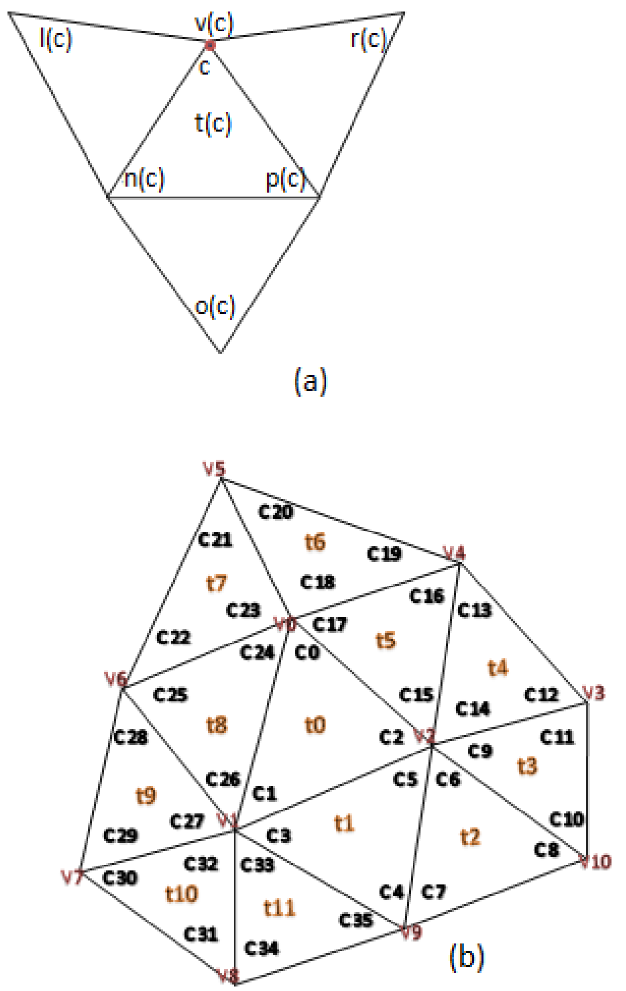
'V[c]' can be constructed by implementing a simple hash table and 'O[c]' can be constructed by identifying all the corners associated with a vertex and revolving around each vertex, marking the opposite corners. For more specific implementation details, refer to the original publication on corner table [9, 10]. Once the required arrays are populated, a simple command called 'swirl' can be easily implemented to identify the number of shells in the model. Refer to [9, 10] for more details.

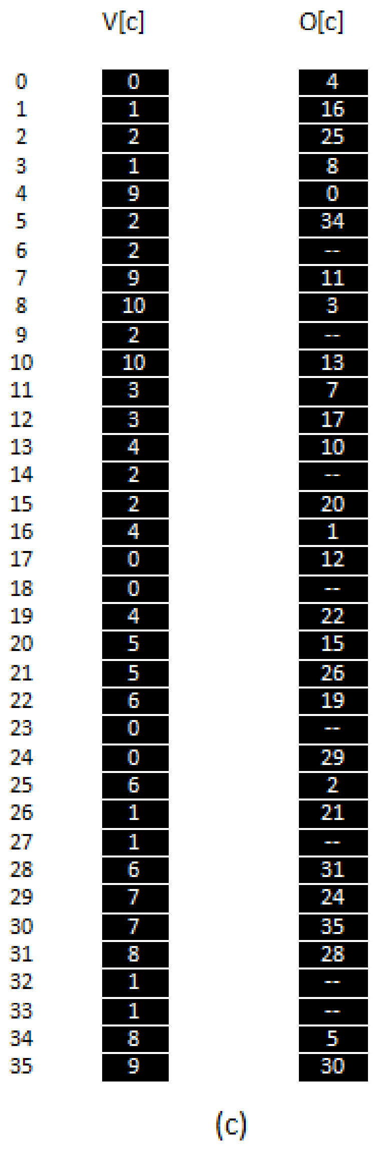
In this way, once all the required topological information is extracted from the STL file, it can easily be translated into a format that ACIS can recognize and work with. An algorithm with these ideas has been implemented and STL files were successfully sliced with the ACIS kernel.

IV) Direct Slicing of STL files:

An algorithm for directly reading and slicing STL files has also been implemented. One of the major issues in efficiently slicing an STL file is being able to quickly identify those facets that lie in the intersection region from the rest of the facets in the file. Tata et al. proposed a facet grouping strategy for this purpose [11]. Luo et al. proposed yet another strategy for identifying these facets quickly [12]. The algorithm presented here is very similar to theirs with minor modifications for speed improvements and to suit the needs of LAMP process. The schematic in

Fig. 4 illustrates the procedure for identifying the intersecting facets. In order to identify the intersecting facets, first each facet's maximum and minimum z-coordinates are computed and stored in memory (slicing assumed to be along the z-direction). Then, for a given slicing plane, first all the facets whose minimum z-coordinate is lesser than the slice plane height are selected (fig. 4(b)). Out of these selected facets, only those whose maximum z-coordinate is greater the slice plane height are identified and retained while the rest are discarded. This way, only those facets that intersect with the given slicing plane are isolated from the rest of the facets in the file (fig.4(c)).





	V[c]	O[c]
0	0	4
1	1	16
2	2	25
3	1	8
4	9	0
5	2	34
6	2	--
7	9	11
8	10	3
9	2	--
10	10	13
11	3	7
12	3	17
13	4	10
14	2	--
15	2	20
16	4	1
17	0	12
18	0	--
19	4	22
20	5	15
21	5	26
22	6	19
23	0	--
24	0	29
25	6	2
26	1	21
27	1	--
28	6	31
29	7	24
30	7	35
31	8	28
32	1	--
33	1	--
34	8	5
35	9	30

(c)

Fig.3 : Corner table data Structure: (a) Nomenclature. (b) sample triangular mesh with indexed corners, vertices and triangles. (c) Vertex table "V[c]" and Opposites Table "O[c]". (Note: in a manifold triangular mesh, there will not be any empty cells in the O[c] array as shown here. This is the case here because the array has been populated for a small portion of the mesh only).

In order to do this, a data structure consisting of linked lists is implemented. It consists of a primary linked list sorted in the increasing order of z-values. Each node in this linked list consists of its specific z-value and a pointer to a secondary list that contains all facets with the same minimum z-coordinate value as the z-value of that node. Once all the facets in the given STL file are populated in this data structure, it is straightforward to implement the rest of the operations required to accomplish the steps depicted in figure 4.

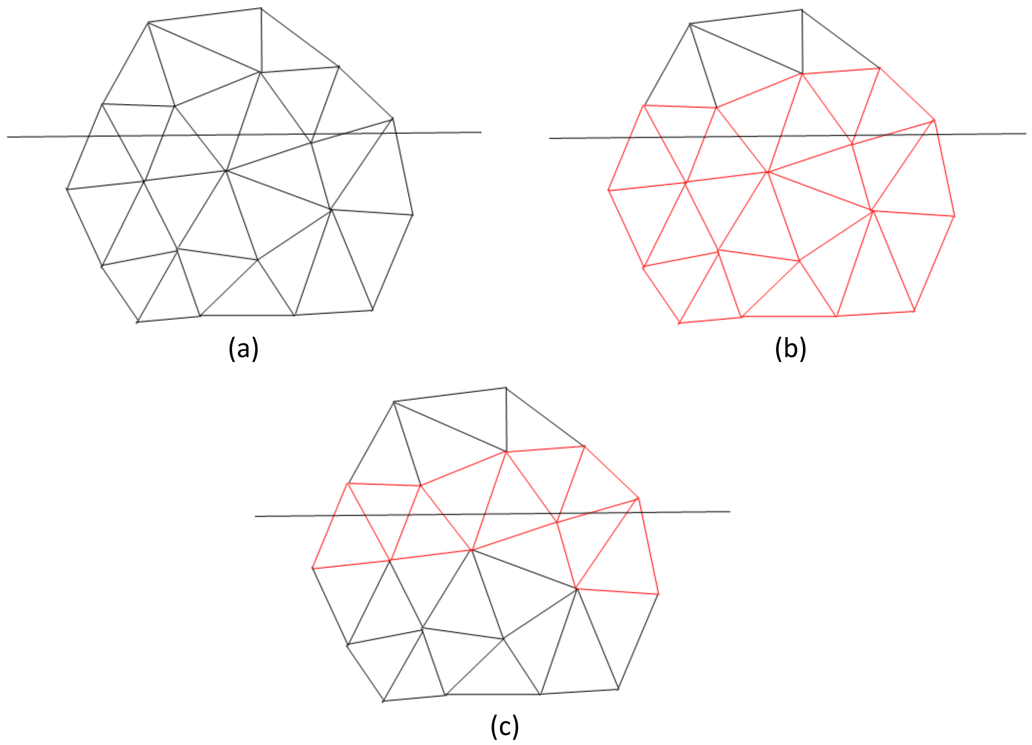


Fig. 4: Direct slicing of STL files: (a) Sample triangular mesh and slicing plane. (b) Facets with minimum z-coordinate lesser than the slice plane height selected. (c) Of the facets selected in (b), only those facets with maximum z-coordinate greater than the slice plane height are kept and the rest discarded.

Once the facets in the intersection region are identified, a simple parametric intersection is computed between the facets and the slice plane to yield the various edges of the intersection wire. Since the facets are listed in a random order, the wire edges are also computed in a random order. In conventional contour planning operations, these edges need to be sorted and the intersection loops need to be constructed. But for LAMP process, it is sufficient to generate a bitmap image of the slice. This can be directly accomplished by shooting rays for each row of pixels and computing intersection points and then using these points, the pixel values could be filled. This process of rasterizing the intersection wire is exactly the same as the one discussed in section I. The images thus obtained are saved in CCITT fax4 format and sent to the DMD chip for exposure.

Results

Various data processing strategies for the LAMP technique have been investigated. A direct slicing algorithm has been implemented using ACIS kernel. This is the most preferred way of generating slice data for LAMP process because it doesn’t have the inherent inaccuracies associated with an STL file. It is an efficient and accurate method to generate slice data and is especially suitable for LAMP process owing to the high feature resolutions. But through this approach, we can only handle specific CAD file formats ("SAT" file format or an equivalent format that is translatable to SAT). Since STL files are the defacto industry standard, several methods to process STL files have also been investigated. Slicing using POVRAY is a simple and elegant way to produce the data as it directly renders the bitmaps needed for LAMP process thereby alleviating all the intermediate steps that could lead to inaccuracies. But this approach is inherently slow owing to the underlying computationally intensive ray-tracing process. An alternate method to use ACIS for slicing STL files by reconstructing the topology information was investigated. An efficient data structure for extracting the topology information was implemented and the STL files were successfully translated to SAT files which were then processed using ACIS. This holds immense potential for implementing various STL file manipulation and error corrections routines using the robust inbuilt functionalities of ACIS. But the inherent drawback of this approach is that the resulting SAT files can be extremely large (several gigabytes) for complex STL files, because of the millions of entities in terms of vertices, edges and facets in the resulting CAD model and such enormous SAT files can be difficult to handle. Finally, to overcome these limitations, an efficient algorithm for directly reading and slicing STL files has been implemented. In conclusion, the ACIS approach was best suitable for direct slicing of CAD parts while the algorithm for directly reading and slicing triangular meshes yielded best results for STL files.

References

1. Guduri S, Crawford RH, Beaman JJ, "A method to generate exact contour files for solid freeform fabrication", Proceedings of the Solid Freeform Fabrication Symposium, Austin, Texas, pp 95–101, August 1992.
2. Vuyyuru P, Kirschman CF, Fadel GM, Bagchi A, Jara-Almonte, " A NURBS-based approach for rapid product realization", Proceedings of the 5th International Conference of Rapid Prototyping, Dayton, Ohio, pp 229–239, June 1994.
3. X. Chen, C. Wang, X. Ye, Y. Xiao and S. Huang, "Direct slicing from PowerSHAPE models for rapid prototyping", Int. J. Advanced Manufacturing Technology, 17(7), pp. 543–547, 2001.
4. W. Cao, Y. Miyamoto, "Direct Slicing from AutoCAD Solid Models for Rapid Prototyping", International journal of Advanced Manufacturing Technology, 21(10), pp. 739-742, July 2003.
5. Stephen J. Rock, Michael J. Wozny, “Generating Topological Information from a bucket of facets”, Proceedings of the Solid Freeform Fabrication Symposium, Austin, Texas, 1992.
6. H. Lopes, and G. Tavares, “Structural operators for modeling 3-manifolds”, Proc. ACM symposium on Solid Modeling and Applications (SMA), ACM press 10-18, 1997.
7. B. Baumgart, “Winged Edge Polyhedron Representation”, AIM-79, Stanford University Report STAN-CS-320, 1972.
8. M. Kallmann and D. Thalmann, “Star-vertices: a compact representation for planar meshes with adjacency information, Journal of Graphics Tools, 6(1), p.7-18, Sept. 2001.
9. J. Rossignac, A. Safonova and A. Szymczak “Edgebreaker on a Corner Table: A simple technique for representing and compressing triangulated surfaces,” Hierarchical and Geometrical Methods in Scientific Visualization pp. 41- 50, 2003.
10. J. Rossignac, "Solid and Physical Modeling", Chapter in the Wiley Encyclopedia of Electrical and Electronics Engineering. Ed. J. Webster, 2007.
11. Kamesh Tata, G. F., Amit Bagchi, Nadim Aziz (1998). "Efficient Slicing for Layered Manufacturing", Rapid Prototyping Journal 4(4), p. 151-167.
12. Luo, R. C., P.-T. Yu, et al. "Efficient 3D CAD model slicing for Rapid Prototyping manufacturing systems", IECON proceedings San Jose, CA, USA, IEEE, 1999.