Modeling and Visualization of Three-Dimensional Scenes

F. Nunes Ferreira DEEC.FEUP/INESC.Norte
A. Augusto Sousa DEEC.FEUP/INESC.Norte
V. Afonso Branco DI.ISEP/INESC.Norte
A. Cardoso Costa INESC.Norte

First Luso-German Meeting on Computer Graphics
Lisbon, October 1988

ABSTRACT

A small group of the Computer Graphics & CAD team from INESC.North has dedicated a large part of its time effort in investigating "Image Synthesis with Large Level of Realism".

Initially, studies involving the Acquisition, Modeling and Visualization of three dimensional scenes were carried out. Results obtained in this first phase were a solid modeller, and some versions of visibility algorithms.

The modeller, which has been repeatedly improved, already exists as a functional version for a UNIX workstation. It has been installed with locally adapted algorithms, which permit Ray-Tracing and Animation.

The generation of surfaces through "Generalized Revolution", based on interpolation curves involving some original ideas, will soon be included.

The Visibility Algorithms are still a target objective within our work. Some new ideas developed by the group in this area are now being explored.

The working objectives of this group are:

-the acquisition and modeling of three dimensional scenes
-visualization with large level of realism

In the first case, we will be handled by image processing techniques; in the second, we will explore a set of new ideas in the area of Special Architecture for Computer Graphics, using the available technology as a base, but not putting aside investigation in the domain of VLSI technology for Computer Graphics.

Themes to be developed:

-Solid Modelling
-Surface Generation through "Generalized Revolution"
-Visibility

SOLID MODELLING

The system MS(E) (Experimental Solid Modeler) has been developed to become a work bench for teaching 3D Computer Graphics and Solid Modelling.

In 1985 we began our work in the development of a system that should support the algorithms for hidden line/surface removal [Sou87] and also the study of (simple, natural) interactive ways for model description acquisition.

Our first prototype, GAUDI1.0/86 (Macintosh 512 + UCSD Pascal), has been an important step, because it included already the capability of constructing solid models by "rubberband" translational sweeping, operated from a flat figure obtained with a 2D Drawing editor developed locally; the 3D geometric nucleus was very poor limiting the complexity of the models.

The second version, GAUDI2.0/86/87, written in TML Pascal for the Macintosh Plus, surpassed some problems of early version, first in the 3D geometric nucleus and second, in the interactive operators for models construction. In this version we have implemented the capability to operate the translational sweep and also the rotational sweep and the capability to individually modify spatial positions of vertices. A hidden line removal algorithm has been added, complementing the existing hidden surface removal from the first version.

At last, system MS(E), written in C and ListPack (Carnegie-Mellon) on a workstation HP9000/320, became with a better version of GAUDI2.0 geometric nucleus and with the modules to perform the calculations of geometric transformations of models, movement modelling [CoB88] and ray tracing [AlB88].

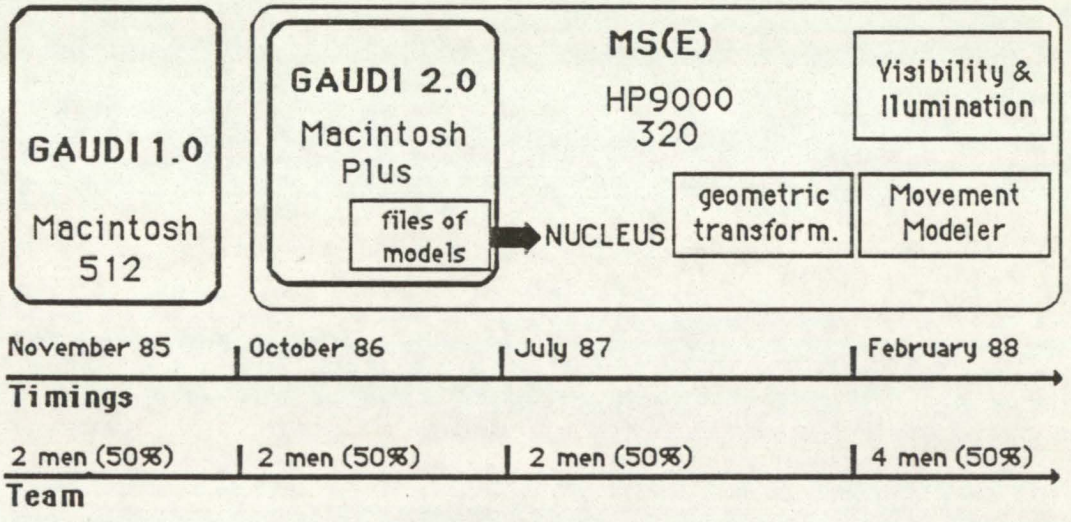

fig.1 - System MS(E) Evolution

Geometric Nucleus and 3D Models Construction

B-REP (Boundary-Representation) has been the representation scheme elected to describe the solids models (fig. 2). Solids are represented by the set of their faces, edges and vertices. This kind of representation is not ambiguous and can be used as an information source for other applications (geometric properties evaluation, interference analysis, etc).

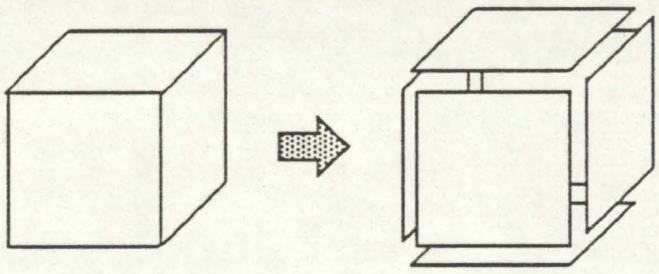

fig.2 - B-REP Representation

We have used Euler Operators [Bau75]. They are atomic actions that enable the production and modification of 3D models, assuring their topological validity. Euler Operators are based in the extended Euler formula, which relates topological components of an object representation.

$$v - e + f = 2 * (s - h) + r$$

(**v**) (**f**) (**e**) - number of vertices, faces and edges
(**s**) - number of shells
(**h**) - number of holes
(**r**) - number of rings

A complex object is normally an assembly of other elementary or less complex objects; multiple occurrences of a component are usual. The **instance** entity appears as a way to use less memory in the representation of complex objects, because it enables components to be represented only once. Each instance has its own spatial position and orientation, which distinguishes it from other instances associated with the same component.

A complex object became an acyclic directed graph with assemblies in nonterminal nodes, elementary components in terminal nodes and instances in the links.

The figure (fig.3) shows topological hierarchy with the related geometric data. **Winged-Edges** data structure [Bau75] was used to represent lower part of the hierarchy (edge, face, ring, vertex).

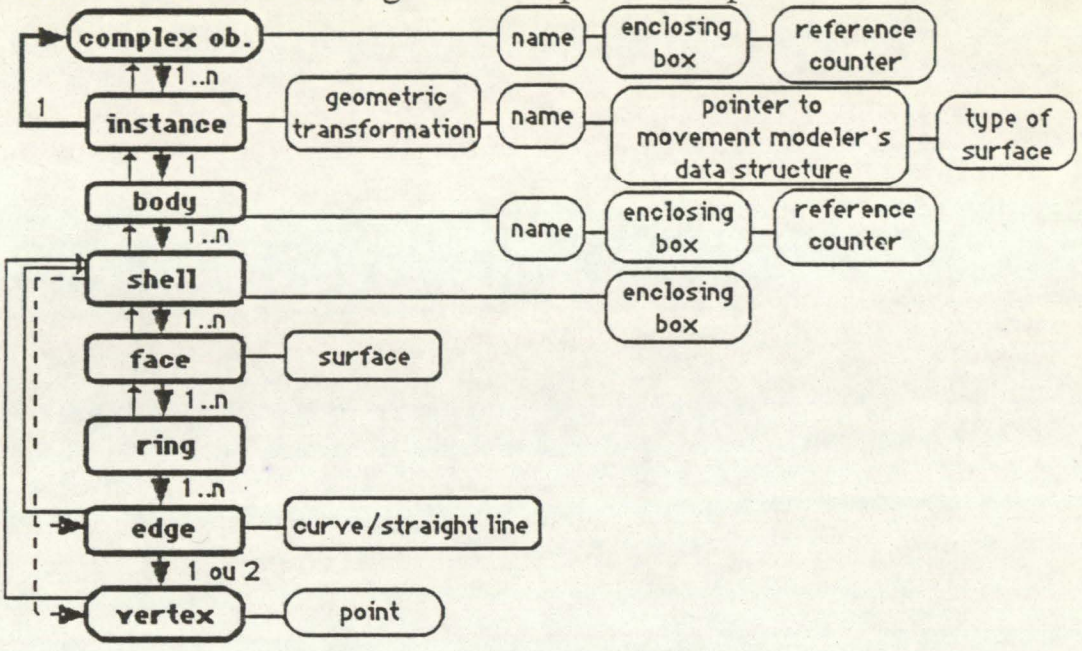

fig.3 - Topological Hierarchy

A 2D drawing is the source for the "automatic" construction (without interactive help from the user) of a "lamina" (3D abstract solid without thickness) with the same topology of the initial drawing, but already a 3D model.

3D construction operators (ex: sweeping operators) will give desired shape to the solid, with the topological or/and geometric modifications to the model.

The bounding box is included in the data structure to allow faster geometric tests, when these need not to be very accurate.

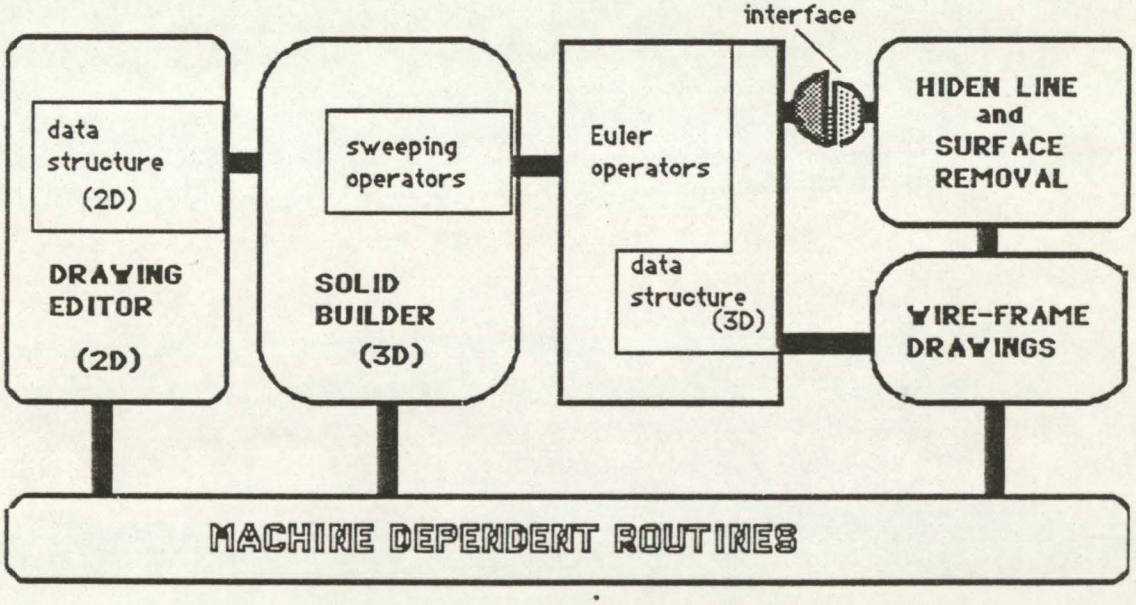

fig.4 - GAUDI 1.0/2.0 Architecture

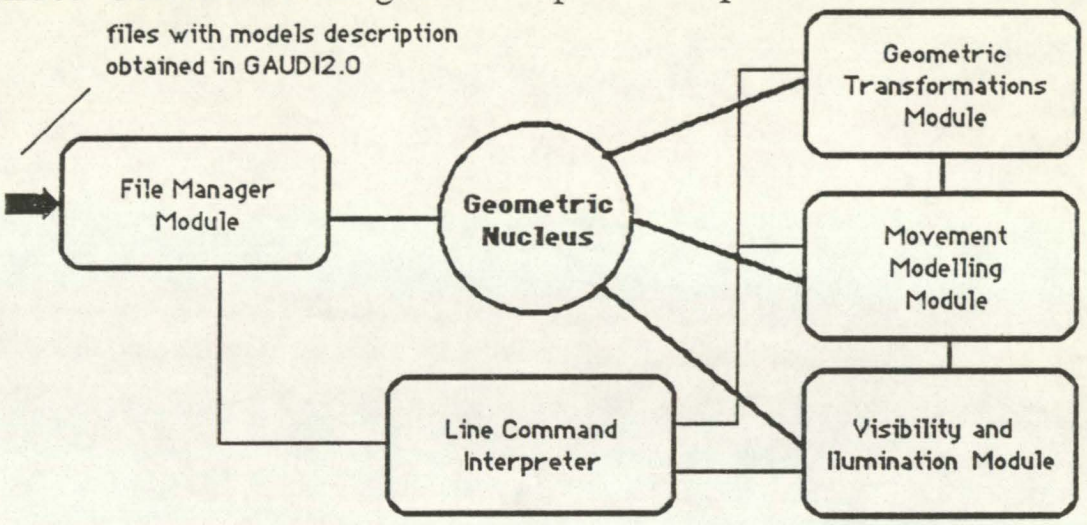

fig.5 - The Architecture of System MS(E)

At the moment, MS(E) system has a command line interpreter as the more common way to interact with the user. Objects names become fundamental object attributes for their identification.

SURFACE GENERATION THROUGH "GENERALIZED REVOLUTION"

The generation of surfaces using a non-standard approach is explained here. The basic concepts involved are:

-the **contour**, a non self-intersecting closed line (can be similar to a **section**, although the **contour** does not have to be planar);

-the **profile**, a non self-intersecting line, joining two different **contours**;

-the **axis**, a non self-intersecting line, joining two different **contours** by its interior.

Any surfaces may be defined using this technique, and their definition is generally simple. See figure 6 (a cylinder).

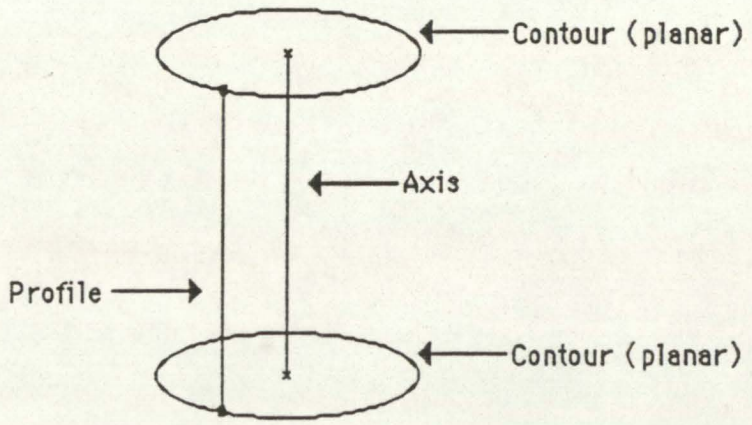

fig. 6- Basic Concepts

A more complicated surface may be defined if the contours are non-planar and if more than one profile is considered (non-straight lines). See figure 7 (a very deformed cylinder).

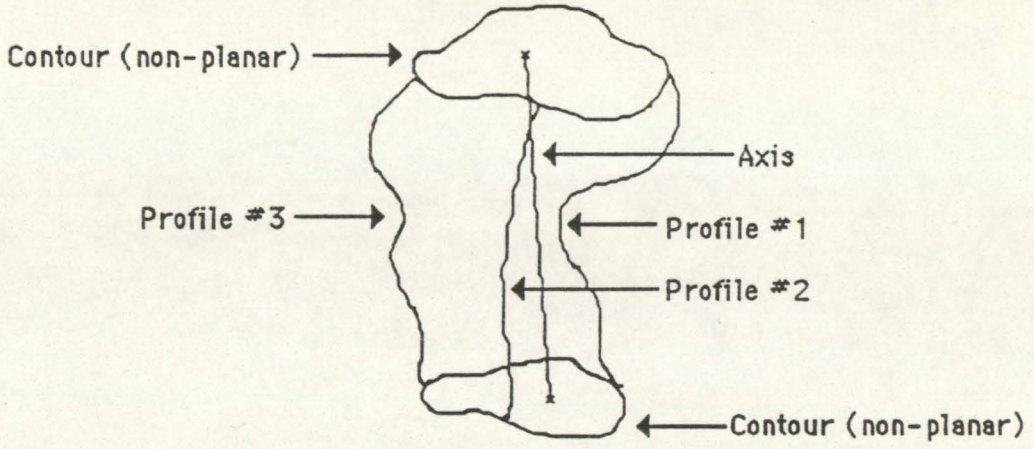

fig.7 - Another Surface (strange)

If one tries to define this surface through spatial revolution, sweeping, etc, it becomes very difficult to accomplish it, because the outer surface is very irregular. Using the concepts mentioned above, the surface is easily defined by:

-1 list of points defining **top contour**;
-3 lists of points defining the **profiles**;
-1 list of points defining **bottom contour**;
-2 points defining the **axis** endpoints.

Implementation

As we are trying to build 3D surfaces, some sort of 3D drawing tool would be desired to define the surface data.

In a 1st phase, the data is input through a text file, that contains the various lists of points needed. The constraints are:

-all of the **contours** are defined using the same number of points (more than 4);

-a **profile** joins a point in a contour to a corresponding point in an adjacent contour;

-between 2 **contours**, all the existing **profiles** must have the same number of defining points (please note that there may be only 1 profile);

-there may be as many **contours** as wanted, each one having a point of interest called *center* (although it may not be the actual center).

(Note: some other rules exist, but they are not as important as these.)

As all the contours are defined by a certain number of points, and because some profiles may be missing (see above), it is necessary to obtain these missing profiles.

The program's main task is to calculate all the profile points that miss. Each pair of contours is taken at a time (let us call it a **cell**), and the all the missing profiles, in that cell, are evaluated, one after the other. When all the cells are exhausted, all the surface data is available (we can view it as an array of points addressed by contour and profile, that is incomplete at start, but the program fills).

The technique employed to obtain profiles from **contours**, **profiles** and **axis** is somewhat complicated; an interpolation is first executed, then iterative improvements are taken until the surface becomes sufficiently *smooth* (fig 8) [FKU77]. The missing points are obtained using data from their neighbor points; as some may not yet be defined, an initial estimate is first attempted. After having these first values, the process is repeated several times, until some program specified error criterion is achieved (this interpolation scheme is rapidly convergent, if the surfaces are not very "irregular").

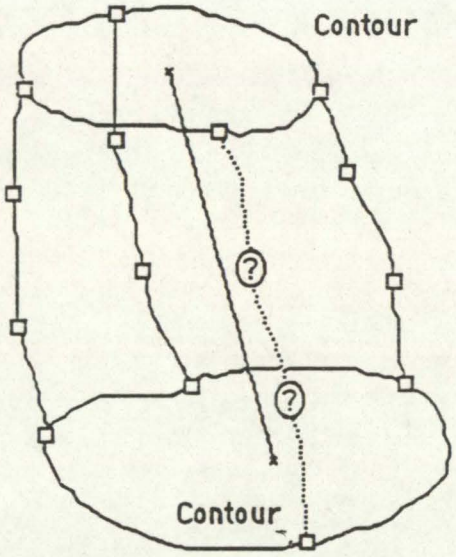

fig.8 - Incomplete Surface. One Profile is missing (2 points)

After obtaining this array of points, it is possible to visualize the surface, using a wire-frame three dimensional projection on a graphic screen (bicubic patches are used to draw the surface, using Overhauser-Coons blending functions for curves and surfaces) [BrA78] [Coo67].

One example can be seen in figure 9. It represents input data (c.), control points (b.) and wire frame output using bicubic surfaces (a.).

The test program is written in VAX-11 Pascal, and uses GKS graphics plus TEKTRONIX 4014 terminal, on a VAX-VMS 4.7 operating system and it is beeing transported to a workstation HP9000/320.

This work shows some aspects and possibilities that are not fully developed. Even so, some results point out that there are very interesting possibilities ahead, and further research will be done in the next future ...

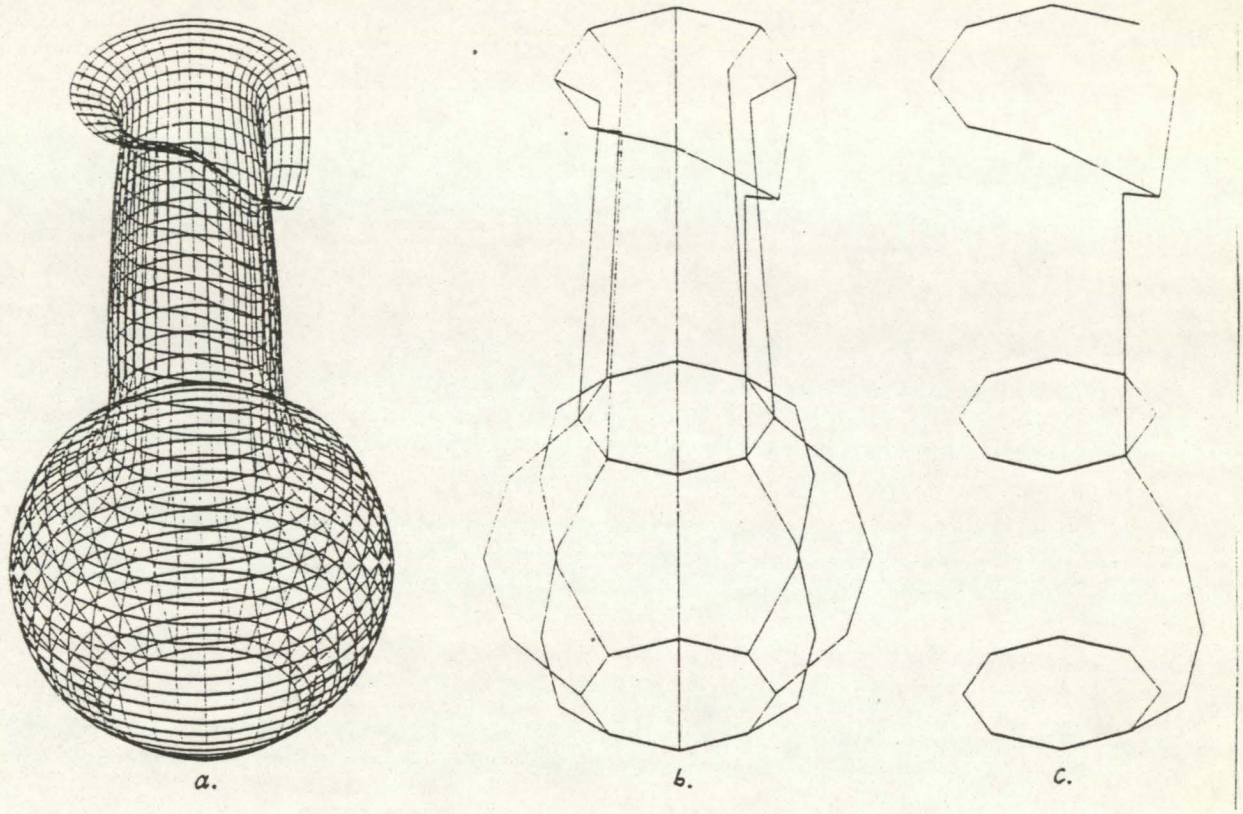

fig. 9 - Surface generated by "Generalized Revolution"

VISIBILITY

Scan Line Algorithms

Objectives

It is known that visibility calculations are basically a big sorting process [SSS74]. Other types of problems in 3D context need sorted data. Then, it is possible to define a structure that can be shared by visibility and other problems.

In the past, we have developed a hidden lines/surfaces removal algorithm that has some subsisting problems. With the acquired knowledge in the previous version, it was necessary to define some structures and methods to solve them.

So, a new dynamic Data Structure, with the enough flexibility to allow the evaluation of new ideas, is being defined. With this structure, we are working on two algorithms, to remove hidden lines and to remove hidden surfaces. In the future, we will obtain more information from the structure.

Data Structure

The data structure is being defined with one main target: to store the most of information, enabling the rest of operations to be simpler and faster. A set of three types of lists maintains in memory all the sorted elements (fig. 10).

VSL- "Visibility Structure List"
List of pointers and Y values that makes possible the access to other lists. It corresponds to Y-sorting.

cAEL- "completed Active Edges List"
List containing all active edges in a particular scan line. It corresponds to X-sorting.

AFL- "Active Faces List"
List containing the faces that are active at the right of the edge pointing to it. It corresponds to Z-sorting.

As can be seen in figure 10, pointers between elements in two contiguous cAEL have been implemented. Each pointer connects two elements in different cAEL but referring the same edge. When the structure becomes completed and the visibility algorithm "travels" along an edge from its upper vertex to its bottom vertex, these pointers are very helpful.

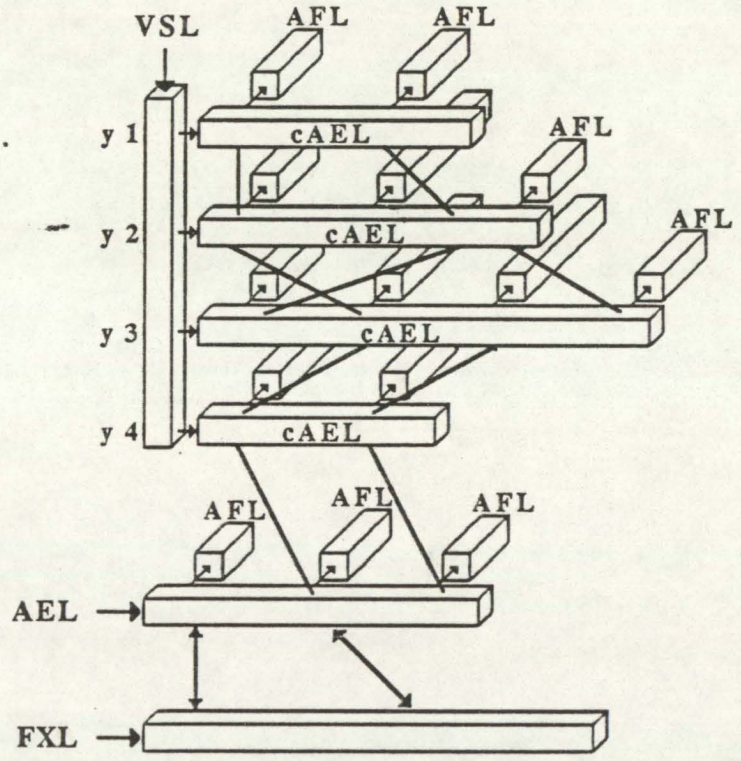

fig.10 - Data Structure

Two more lists are represented in figure 10:

AEL- "Active Edges List"

List containing active edges in a particular scan line. It is different from cAEL in the sense that AEL is still being updated and cAEL is a fully stable list. Actually, each cAEL is a copy of AEL, made in a particular moment.

FXL- "Future Crossing List"

Each element of this list contains a description of an intersection between two adjacent edges in AEL. It is maintained sorted by the coordinates Y,X of the intersections.

Building Data Structure

A raster scan algorithm to remove hidden surfaces was developed. It also was able to remove hidden lines, but produced bad drawings when a nearly horizontal edge intersected other edge and, in consequence, its visible/invisible state was reversed [Sou87].

The problem was that the algorithm worked with discrete values of Y: for each integer value of Y, the AEL was updated by *removing* the edges that became not active, *sorting* the resting ones and *inserting* the new active edges. If, between contiguous scan lines, one edge intersected two or more edges (in the XY plane), the algorithm was not able to correctly identify all intersections and, some visible/invisible or invisible/visible switches were ignored.

Sequim refers the Cross-Algorithm which defines events [SeW85]. An event may be a vertex, an intersection between two edges or a scan line. The Active Edge List is not updated scan line to scan line, but only when the algorithm founds a vertex or a intersection. Furthermore, it is not necessary to update the entire list, but only the event vicinity.

This method can solve our drawing problem, because intersections are evaluated one at a time.

The algorithm to build data structure uses the first two kinds of events: vertices and intersections. The scan line event is implied and makes the AEL to be copied to a new cAEL.

Events must be processed in an ordered sequence, so there is an event detector which uses two Y-sorted lists:

FXL- "Future Cross List"
(described above);

VDL- "Vertices Descriptors List"

Each element in this list is a vertex descriptor, containing:

-the vertex coordinates;
-a list of edges becoming active in the vertex (list **StartEdges**);
-a list of edges becoming inactive in the vertex (list **EndEdges**);

Vertex Processing:

All edges that become inactive in the vertex (list EndEdges) are removed from the AEL; each element in EndEdges points to an edge descriptor and this one points to the corresponding element in AEL to be removed.

If the vertex has no ending edges, then a search is necessary to locate it in the AEL, preparing the next step.

Now, the edges becoming active for the first time (list StartEdges) are inserted in the same location in AEL. In pre-processing time, the list StartEdges must have been sorted by XY slope.

Since these new edges share the initial vertex, they are divergent and, there is no possibility of intersecting each other. However, the first (more left) and the last (more right) new edges will perhaps intersect the old edges at their left and right respectively. These two possibilities must be evaluated and, if intersections are found, they must be inserted in the list FXL, by YX order.

Intersection Processing:

The two edges must be swapped and, as in the vertex processing, they must be tested against their neighbours at left and right respectively, evaluating future intersections.

Output Production

With the structure just described, we can produce two kinds of outputs: Hidden Surfaces and Hidden Lines.

The Hidden Surfaces procedure reads all cAEL, one at a time and converts each one to descriptors of visible faces segments (a visible face segment is a scan line segment, limited by two edges, in which the visible surface is the same).

A visible span descriptor has information about:

-value X of left limit of the span: copied from coordinates of the edge at left of the span;
-visible surface in the span: the first surface in the list AFL pointed by one cAEL element; from edge to edge in cAEL and while the visible surface is the same, the visible span does not change;

A list of visible spans is obtained for a scan line corresponding to the cAEL studied. Actually, the visible spans list is valid for a certain number of scan lines because of the vertical coherence. The only thing to update, from line to line, is the coordinates of the beginning of span.

If desired output is a drawing (hidden line removal), the algorithm follows the pointers that refer the same edge in different cAEL's. If, for a given cAEL, an edge has the same surface at its right and at its left, then the edge is invisible in the corresponding scan line and also in the following scan

lines until next cAEL appears. On the contrary, if the two surfaces (right and left) are different, then the edge is visible.

Special care is needed with horizontal edges, because they do not remain stored in cAEL: a horizontal edge become active and inactive in the same scan line so, when AEL is completed, the horizontal edges have disappeared and can not be copied from AEL to cAEL. The solution is to create, when necessary, a special list of edges named Horizontal Edges List (HEL) in parallel with the cAEL. To determine if a horizontal Edge is visible or not, the algorithm verifies the visibility of the two faces connected to the edge: if one of the two faces is visible inside a span, then the horizontal edge is also visible in the same span.

Enhancements to Actual Work

The structure described above is very expensive in terms of memory. The information included in one list cAEL is probably repeated for many cAEL's because the edges that remain unchanged are represented again and again, whenever the algorithm founds a scan line with vertices or intersections.

However, this version is important to test new ideas and to search for special cases that always appear. The final solution will be based in a smaller data structure: it is sufficient to retain events descriptors in memory and to associate them with portions of AEL that are changed (this changed portions must point to active edges and to lists AFL, just like in the actual structure).

With a list of such events and a well defined set of pointers between events, it is possible to reconstruct one list similar to AEL and, based on it, to obtain a list of spans needed by hidden surface removal. The hidden line algorithm will remain almost the same.

A smooth shading module will be added to the final version. Now, we are planning to implement modules to solve questions like volume determination, cuts by (particular) planes and zooms.

Ray-Tracing

Some work on Ray-Tracing has been made. We hope to continue and to complement this theme with the inclusion of Radiosity [CGr85][GTG84][ICG85].

The algorithm used in Ray Tracing of MS(E) scenes was adapted from [Kuc87], attending the differences in the object representations.

The use of enclosing boxes for simple objects, assemblies and scene, accelerated the process of intersection calculations in two ways:

Reduction in the number of rays

The projection of scene enclosing box allows the reduction of the number of rays to a area normally less than the frame buffer. The background colour is assigned to all pixels outside that area. In scenes with scattered objects, the objects enclosing boxes are useful to decrease once more the action of the algorithm (fig. 11).

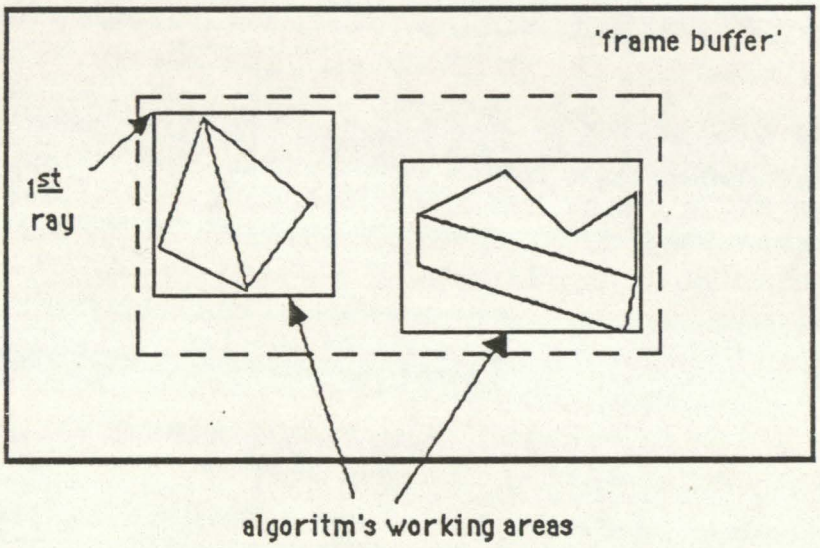

fig.11 - Use of enclosing boxes

Object coherence in the intersections tests

Taking advantage of the available data in the MS(E) topological hierarchy, it is possible to make the intersection tests against the object enclosing box (6 polygons) and then, if necessary, to make the intersection test against all polygons used in the object description (fig. 12).

This kind of test becomes more advantageous as the complexity of objects increases.

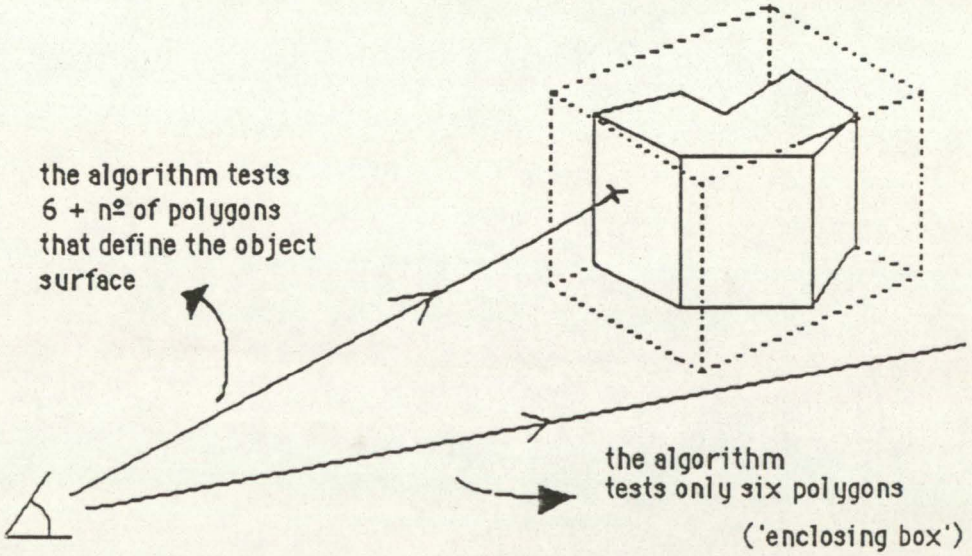

fig.12 - Intersection test against an object enclosing box

CONCLUSIONS

The development of system MS(E) will continue to explore new ways for editing and constructing solid models: Boolean operators, model construction from orthographic views, sketch recognition are some of the possible clues.

Other developments will include the possibility of using different solids representations (ex: analytical descriptions). So, it will be necessary to develop interfaces allowing transformations from one representation scheme to another, or/and to develop different algorithms for each kind of representation.

Until now, the works we have been developing in the rendering area are basically hidden surface/line algorithms and they are justified by interactive applications. These ones need typically short CPU times, not possible with other rendering methods more sophisticated.

Ray Tracing is a very popular rendering procedure and it still is a good theme to research and develop. A new rendering method, named Radiosity, has being discussed and it seems to be able to render very realistic images, by itself or as a pre-processing to Ray Tracing.

Ray Tracing and Radiosity are CPU-time consuming procedures so, they are not adequate to interactive applications. We think that some research can be done in both cases, by adapting known algorithms and/or by developing new ones, using parallel processing capabilities and consequently to reduce CPU-times.

To implement new algorithms, we think that Transputer is a very promising technology [INM87]. Transputers speed in calculations, added to their modularity and ability to do distributed processing, are important attributes which can be very helpful for using ray tracing and radiosity in interactive applications.

REFERENCES

[AlB88] J. E. Almeida, V. A. Branco
Adaptação de um algoritmo de 'ray-tracing' ao sistema MS(E)
1º Encontro de Computação Gráfica, Lisboa 1988

[Bau75] B. Baumgart,
A polyhedron Representation for Computer Vision
AFIPS Proc., Vol.44, 1975

[BrA78] Brewer J. A., Anderson D. C.
Visual Interaction with Overhauser Curves and Surfaces
Computer Graphics SIGGRAPH ACM, 1978

[Bra85] I. C. Braid,
Geometric modelling
Notas preparadas para um tutorial da conferência EUROGRAPHIC'S 85

[Bra87] V. A. Branco,
Modelação de sólidos: síntese de conceitos, realização de um protótipo
Dissertação de Mestrado, Universidade do Porto 1987

[BSF87] V. A. Branco, A. A.Sousa, F. N. Ferreira
Modelação e visualização de objectos tridimensionais
1as Jornadas de PPPAC da Ordem dos Engenheiros, Lisboa, 1987

[CGr85] Michael Cohen, Donald Greenberg
The Hemi-Cube: A Radiosity Solution for Complex Environments
ACM Computer Graphics, Vol 19, N. 3, July 1985

[CoB88] R. F. Costa, V. A. Branco
Modelação de movimento no sistema MS(E)
1º Encontro de Computação Gráfica, Lisboa 1988

[Coo67] Coons S. A.
Surfaces for Computer-aided Design of Space Forms
Project MAC, MIT, June 1967

[FKU77] Fuchs H., Keldem Z. M., Uselton S. P.
Optimal Surface Reconstruction from Planar Contours
CACM, Vol. 20, nº 10, October 1977

[GTG84] Cindy Goral, Kenneth Torrance, D. Greenberg, Bennett Battaile
Modeling Interaction of Light Between Diffuse Surfaces
ACM Computer Graphics, Vol 18, N. 3, July 1984

[ICG85] David Immel, Michael Cohen, Donald Greenberg
A Radiosity Method for Non-Difuse Environments
ACM Computer Graphics, Vol 20, N. 4, July 1986

[INM87] *The Transputer Family*
INMOS Product Information, 1987

[SeW85] Carlo H. Sequim, Paul R. Wensley
Visible Feature Return at Object Resolution
IEEE CG&A, Vol 5, N. 5, May 1985

[Sou87] António Augusto de Sousa

Cálculo de Visibilidade em Cenas 3D:

Síntese de Conceitos, Implementação de um Algoritmo

Provas de Aptidão Pedagógica e Capacidade Científica-FEUP, Porto, 1987

[SSS74] I. E. Sutherland, R. F. Sproull, R. A. Shumacker

A Characterization of Ten Hidden-Surface Algoritms

ACM Computer Surveys, vol 6, N. 1, March 1974

[WCG87] John Wallace, Michael Cohen, Donald Greenberg

A Two-Pass Solution to the Rendering Equation:

A Syntesis of Ray Tracing and Radiosity Methods

ACM Computer Graphics, Vol 21, N. 4, July 1987