Functional Space: An Approach to Build Usable Virtual Worlds from CAD MegaModels

Yann Argotti, Jean Louis Pajon

Abstract

Voluminous CAD models are produced in industry and architecture to design complex mechanical and architectural structures. The use of these models to set up virtual worlds is a rather tedious and laborious work. To simplify this task, we have combined a series of tools which help us to build in a rapid and efficient way virtual world experiments from complex CAD models. These tools perform various operations such as defining the objects which populate the scene, grouping similar objects in categories, giving shape, physical and functional attributes to these categories, defining activities by a set of tasks to perform and a set of means to use, reducing graphics computations by various techniques: use of levels of detail and of culling methods. The resulting virtual world can be visualized in various interactive ways.

1 Introduction

Our work has been done in the double context of our collaboration with the engineering company TECHNIP and of a research project on 3D graphics toolkits and standardization of description languages for virtual worlds.

1.1 Collaborative work with TECHNIP

TECHNIP, which is the leading French engineering company, builds huge oil refineries in various parts of the world. This company has to answer the following question: how to replace real scale models by virtual ones ? Before actually building such a plant, a model at 1/30 scale is made in Paris. Such a model can take six months to be built and can be larger than 10×10 meters as the actual oil refinery can extend over 1×2.5 kilometers. These scale models are necessary to check various spatial and ergonomical properties such as accessibility and transportability.

1.2 Standardization Issues For Building and Exploring Virtual Worlds

A lot of efforts are made in order to standardize the building and exploration of interactive virtual worlds.

Techniques in 3D graphics are well known. They are now available in some standard graphics libraries. Futhermore, the visual features of virtual worlds can be described in some description languages like VRML which are becoming more and more common. Such file formats integrate a lot of non graphical features, sounds, etc. Some are particularly adapted to specific activity domains such as the Flight format which is very appropriate for simulators.

In the same time, a lot of work is being done in the field of communication protocols such as DIS for interconnecting people in virtual worlds.

The success of the Web leads also to various efforts in standardization of file formats for multimedia documents and of protocols for transfering them over the Internet, possibly in real time.

There is a need to integrate these competing or complementary standardization efforts into an unified model. In this context, we have combined the use of various tools which are aimed at providing the best compromise between efficienty and standardization.

1.3 Outline of this Work

This work will be described in several parts. We begin by presenting a platform to experiment virtual reality concepts. We then suggest a general model for describing what a virtual world is and how participants can exploit and use this world. Some specific techniques to extract specific objects from the scene and to optimize the visual rendering of this world have been used. Finally, we describe some examples of activities made possible in virtual oil refineries and then some implementation issues with commercial software packages (Open Inventor, Clovis).

2 A platform to experiment virtual reality concepts

2.1 General Framework

To conduct virtual experiments, several steps can be defined and numerous tools can be involved at each step. It is common to distinguish three basic steps:

- Various types of data must be defined to populate the virtual world. Data may consist of geometric objects but also of functional objects, tools, activities, people, etc. The aim of a virtual world must also be precisely stated.
- Data of various origins must be integrated into the scene.
- The scene must be rendered and interactively explored.

A lot of software tools are available to perform specific tasks involved by all these steps.

- Specific and numerous tools are available to model specific data. Sounds and images can be generated by various means. Geometric data can be created by various 3D modelers such as Alias, Multigen, AutoCad, Euclid,

International Conference Graphicon 1998, Moscow, Russia, http://www.graphicon.ru/

Catia and many other tools. Object trajectories can be generated by animators such as Explore of Wavefront, etc. Simulators can be used to provide various data in real time.

- The integration of many types of data into a scene is a difficult task, as it requires both the knowledge of many areas of activity (graphics, physics, behavior, sounds, etc) and their unified treatment inside a global model. This model can be either expressed in some low-level programming languages like C with a graphics library or more effectively in some description language like VRML 2.0 or the Flight format of Multigen.
- The exploration of a virtual world can be achieved by various types of software tools:
  - If a programming language such as WorldToolkit or Performer is used to express the mechanism of a virtual world, this language can be either compiled or interpreted.
  - If a file format like Open Inventor or Flight is used we only need an appropriate browser which is able to read such formats. This browser can include a language interpretor if scripts written in programming languages can be embedded in a description language format like VRML.

In our experiments, we suppose that most data have already been generated in an initial step.

Step 2 is performed in several stages:

- CAD models are converted in Open Inventor format
- geometric and management operations on graphical databases are performed with Open Inventor library
- data are integrated with a C program in a description file format which is more appropriate for our purposes
- this file format is then translated into Clovis code and geometric data in Wavefront file format

At the present time, step 3 is often performed by Clovis browser [4,5] which is able to interpret scripts written in a specific language and which we are going to described soonly. Occasionnaly, we use also somw viewers of SGI like perfly in Performer for large models and Open Inventor tools (ivview, gview, etc) for small models and various VRML browsers: i3d [1], webview [11], vrweb [8], etc.

2.2 Clovis: a powerful browser for interactive virtual worlds

To offer practical solutions for exploring large and complex virtual worlds with current technology, some prerequisites must be fullfilled. Apart from the graphics hardware which must be able to render a lot of polygons at interactive frame rates, several considerations have guided us in the choice of software tools:

- Instead of writing these tools from scratch for this particular application, it could be interesting to use an existing software package which was able to offer a partial solution to our initial problem, without any programming.
- This software package must not be limited to navigation tasks. It must allow us to specify various interaction tasks.
- To specify particular interaction tasks, this software package must be flexible enough in order to provide as many new features as could be wished. It was necessary to be able to augment its capabilities as the uses of a virtual are numerous and cannot always be defined in advance.

For these reasons, we have chosen to use Clovis software package of the French company Medialab [5,6] because it has both the ability to display various types of graphical data (geometry, materials, textures, light sources, objects trajectories, etc) and to interpret programming scripts aimed at specifying various navigational and interaction tasks.

Clovis has been experienced with 1/20 of an oil refinery, which consists of 650 objects made of 450 000 polygons.

The main of Clovis for industrial applications is to give a personal experience to an user which has to deal with a virtual environment through various real and virtual tools. To set up this experiment, several steps can be distinguished.

In a first step, the experiment we want to achieve is precisely described. To perform this task, we obviously specify the objects which constitute the scene and their properties (materials, etc). But we have also at our disposal a flexible language analogous to a subpart of C++ and a set of about 300 functions. It is possible to write programming scripts either directly in this language or indirectly by using "script generators" which transform some simple commands into scripts. Such generators are mainly used to perform usual tasks such a management of several universes and simple grabbing and displacement tasks. For example, it is possible, as in dVise of DIVISION, to specify the degrees of freedom with which an object can be displaced (x, y, z translations or rotations) with some joystick for example.

In a second step, before actually performing the final experiment but after having launched Clovis with the files previously written in step 1, we can interactively change various parameters through a 2D-interface (which can also be enriched in the first step). For example, materials can be modified, textures can be precisely positioned, objects can change of materials, locations, etc. This step allows in fact to update the 3D scene but not the strategies used to interact with it.

In the final step, after a specific 3D scene has been set up, it can be fully experienced in real time with the various methods described in the programming scripts. It is important to notice that several methods or configurations can be defined, and that the 2D-interface allows to choose between them. For example, the same universe can be explored either with a keyboard interface for non-immersive experiments or with 6 DOF-trackers for immersive walkthroughs.

2.3 Scene Generation: the use of Open Inventor to manipulate scene graphs

The Open Inventor graphics library is a very rich toolkit which allows to perform various manipulations on hierarchical graphics databases called scene graphs. We used this library to perform the various operations on the data which populate the world:

- Splitting of CAD files into as many files as there are objects we want to distinguish in the final Clovis experiment.
- Geometry simplification of each object in order to generate several levels of detail
- Cleaning of defectuous polygons and generation of normals.
- Conversion from Open Inventor format to obj or geo formats of Wavefront.

After objects have been generated and organized in the final scene, various other data from numerous origins must be integrated into the global model. In particular, this model must take into account the activities which are aimed at performing various tasks with specific sets of tools and the people who practice these activities.

3 Functional Space: a description language for repetitive virtual models

We describe here some characteristics of a description language for virtual worlds which contain a lot of repetitive objects. It is called 'Functional Space' to express the fact that a virtual space can be of some usefulness only if it is inhabited by objects which possess functionalities as in the real world. Before defining more complex aspects of the world, physical appeareances and fonctionnalities of objects allow to define various scenarios to take part in this world.

This language is built on an underlying object-oriented model of software system for building and controlling a virtual world. Today, this model is only implemented in Functional Space description language but we intend to use it in the library of classes used by a parser for generating virtual worlds which could be used by commercial 3D visualization tools such as Clovis or any VRML browser.

This model includes the following considerations:

- It is important to be able to describe in a simple way the characteristics which are shared by classes of objects. In this way, Functional Space is a compact file format which allows us to abstract common properties into categories. This abstraction mechanism, more practical to use than the DEF instancing method of Open Inventor, is similar to the concepts of prototype in VRML 2.0 and of class in object-oriented languages, but it is rarely found in graphical file formats where all objects are independently defined. It simply reflects the fact that the real world is also full of similar objects, made of many simple possibly articulated components. Catalogs of objects and of components can be easily defined.
- Geometry is not described in this model. Open Inventor is used for this task. This model is aimed at describing the functionalities rather than the shape of objects.
- We cannot dissociate a virtual world and the tools a software system must possess to control this virtual world. For example, 2D control panels are part of an interactive exploration of a virtual world and must be taken into account in its description.

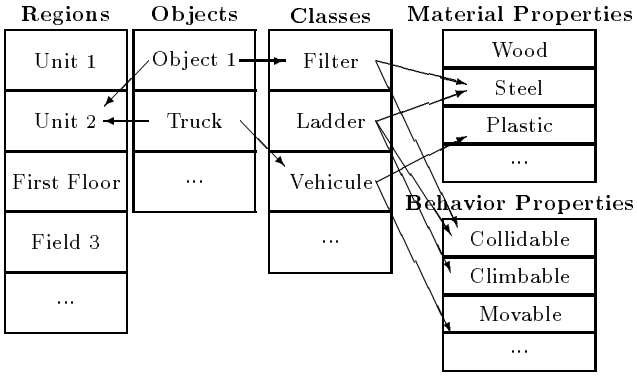


Figure 1 - Example of objects classification in Functional Space

Basically, we consider that a virtual world is made of decorative or functional objects, of activities that can be performed to change aspects of these objects and of people who exert several of these activities during some period of time. These activities imply the use of specific tools that can be some of the objects which populate the world or physical devices, control panels or external programs.

3.1 Space Hierarchy

First of all, we consider a large world which can be splitted into a hierarchy of subdivisions: countries, regions, etc. The user can choose the number of levels and the names given to the levels of this hierarchy. Terminal nodes of this hierarchy can be regularly subdivided into cubic or parallelepipedic cells to allow culling methods.

Nodes can be used to store large portions of the world into disk. Only the specific nodes which are currently displayed can be kept in memory.

Objects are then patitioned into the terminal nodes of the spatial hierarchy.

3.2 Functional objects

We define a functional object as being characterized by the following features:

- shape
- physics
- operability
- use

A decorative object is only defined by its shape.

Let us give some examples of the features of a functional object.

- Shape is given by the combination of these two basic features: a geometry and an appearance which can be animated in time in an arbitrary way (simulation, animation, real-time capture). These two features can be precisely defined by using computer graphics concepts such as the concepts used in the VRML file format. An object can be made of several parts, which are themselves objects. Either these parts are rigidly linked together or some degrees of freedom can be defined.
- A mass can be associated with an object. We can also decide if an object will be considered for collision detection during walkthrough. In the case of an articulated body made of several subparts, the constraints between these subparts must be defined.
- It is difficult to describe the operability of any object, the way it works. Let us give the simplest and most common example: the selection or motion of some subpart can change the aspects of some other parts. For example, this technique allows us to change the direction wheels of a car by moving the steering-wheel.
- Some uses of an object can be easily characterized at this level which does not take into account the final actors of the virtual scene. For example, an object can be used to navigate inside a scene in some specific ways. In a building, staircases or ladders can be climbed up and down. Other types of objects can be used to view this scene in specific ways, as is the case of glasses. To implement these uses, various algorithms must be used. In the VRML 2.0 description language, this requires to write specific scripts in Java for example.

As it can be difficult to separate the physics, operability and use of objects, we consider that they can be integrated to define physical and behavior properties.

Objects can be regrouped into classes in order to associate various characteristics with each class instead of specifying them for every individual object. This approach is performed with the help of classes of properties (Fig.1). We regroup objects in classes and we define properties for each class. In this way, we do not have to define for each object its particular properties. Our parser for Clovis code has then a mechanism

to automate the assignment of properties to all objects of a class.

A simple example can be found in the appendix.

3.3 Activities

We define an activity as as being characterized by a set of tasks to perform and a set of means to perform these tasks. These means comprise:

- 3D Virtual Objects
- 2D Control Panels
- Devices
- External Programs

For example, an external program has been used in our studies for displaying a floor map of an oil refinery on a specific screen along with the 3D representation of the plant on another screen. Standard communication protocols link these two programs.

The most common tasks are the following:

- vision
- motion
- selection

Examples of selection tasks include the grasping of objects with the hook of a crane or the choice of an object with a pencil in order to add comments to this object.

4 Semi-Automatic Extraction of Scene Objects from CAD MegaModels

Extraction of significant objects from CAD models is not an easy task. Basically, we have a three-level hierarchy composed of the universe, objects and geometric primitives (Fig. 2, 3). Primitives are solid shapes such as spheres, cylinders, cubes, etc. Objects are built by instancing and combining these primitives. This operation can be performed with CSG boolean operations. Objects can be regrouped into units or equipments but we will not take such regroupings into account in this study.

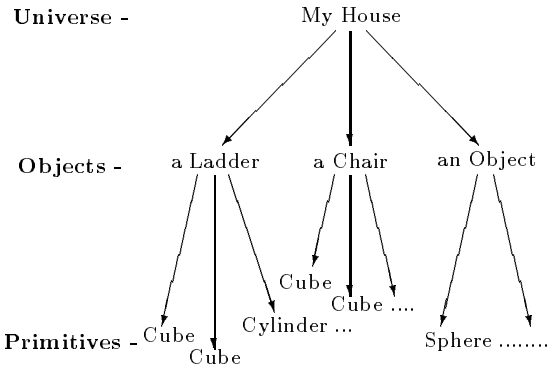


Figure 2 - The three-level hierarchy of a world

To perform specific operations on specific objects, we need to identify them.

Futhermore, in Clovis, separate objects are stored in separate files. There are many interests in having a lot of objects. For example, specific properties like color or texture can be given to an object (but not to one of its subparts).

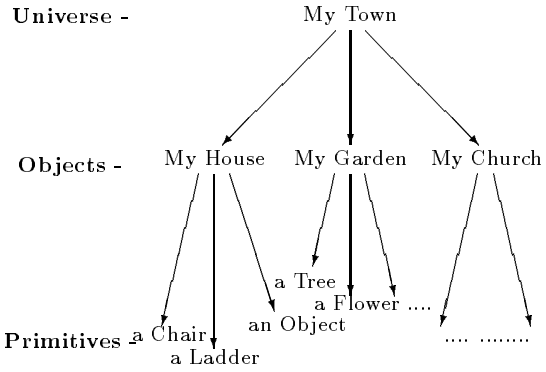


Figure 3 - Another way to define the Hierarchy

Only an object can also be grabbed and displaced during interactive walkthrough with Clovis. Moreover, an internal Clovis feature allows to cull objects which are outside the field of view. To fully exploit this feature, it is important to have many small objects.

The most crude technique to separate Open Inventor scene graphs into myriads of objects is to generate these objects from all the nodes which are located at a specific level. This could give rise to too many objects. Some other cues can be used to distinguish objects in a scene graph. Ideally, each object must be properly designated with some significant name during the world modeling phase which has given rise to DXF and IV files. These names can be stored as labels in a iv file. In this case, it is easy to separate into distinct files all objects which have a specific and meaningful name (which has been stored in a list of meaningful names).

Unfortunately, in many cases this interesting information is missing in CAD files. We cannot restore it but two cues are used to split the CAD file: we distinguish parts of an OpenInventor scene graph which have different materials or which are geometrically separated.

To check this second feature, a simple and not very general stratagem (that will be improved in the future) is used. At a given level of the scene graph, we compute the bounding-boxes of all subsequent nodes. Then we determine all the connected components made by these bounding-boxes.

To be more explicit, let us consider that two bounding-boxes A and B communicate with each other if there is a series of bounding-boxes starting from A and going to B which intersect one another two after two along this chain. A connected component of bounding-boxes is then the greatest set of bounding-boxes which communicate one with another. Such a connected component is considered here as being an object which will be stored in a separate Open Inventor file.

In some circumstances, CAD files can contain a few large objects that cannot be splitted by the previous strategies. Even in this case, it can be useful to partition such an object into several smaller ones. This can be done in various ways. In our case, we simply store in terminal files the contents of the terminal nodes.

5 Graphics Rendering Optimization

To improve frame rates, common stratagems are used, such as culling of back-facing polygons and of objects that are outside the field of view.

In virtual reality applications it is not unusual to define several distinct universes linked by virtual doors and such that when the user is inside one of these universes he sees and he interacts only with the objects which belong to this universe. He can then go to another universe by opening the virtual door which links them.

This idea works well for buildings which contain a lot of closed rooms. This is not the case of oil refineries for example and similar situations where lots of objects are defined in a large and open space.

For these reasons, we have combined the use of some other techniques to reduce the amount of polygons that are drawn. When the user moves, many objects disappear or lose their details. Futhermore, we consider a spatial subdivision of our model to compute cells that are outside the field of view.

5.1 Data Simplification

The geometric representation of distant objects can be simplified. Original models are solid objects represented by polygonal surface boundaries. Resulting simplified models can consist of lines, points, polygonal objects and also textured objects. We have implemented some simplification methods but we let the opportunity to add user defined methods with dynamic shared libraries.

We can distinguish two main methods to simplify data, substitution methods and algorithmic methods.

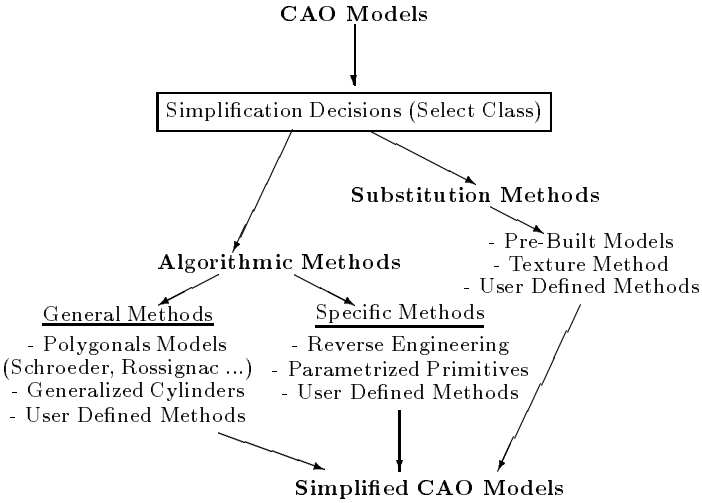


Figure 4 - The Simplification Process

5.1.1 Substitution Methods

These methods take a CAO model and substitute it by another model.

Pre-Built Models In this case, we simply have a catalog of object classes and for each object class we have a set of predefined geometries. These geometries have their proper coordinate system. Let us consider a global coordinate system: each geometry and each object of the scene are positioned in this coordinate system.

To simplify a given scene object at a specific level of detail, we must perform the following tasks:

- Determine its class and select the appropriate geometry
- Compute the proper geometric transformation from this geometry to this object
- Apply this transformation to the given geometry

To obtain specific geometries, various methods are possible but generally they are crafted by hand.

Texture Method In some cases, as for example ladders, fences and other particular objects well adapted, it is possible to define simple geometries such as parallelepipeds and to apply some proper textures on these shapes. We obtain textures by projections of objects on several planes. Moreover, it is possible to simulate the fact that sometimes we can have holes in an object, as for fences or ladders, by using alpha image component. This solution is appropriated to a machine where textures are cabled.

5.1.2 Algorithmic Methods

These methods take a CAO model and work on its vertices and faces to obtain a simplified model.

Reverse Engineering In some particular cases, it is possible to fit a simple geometric shape to a given set of points. For example, if we know that a given shape is a cylinder, it is possible to compute the characteristics of this cylinder (position, rotation, scaling) from these points. The scaling factor gives us the height and radius of the cylinder. Then we can perform simplified cylinder with tranformation informations.

Parametrized Primitives Parametrized primitives are solid objects defined in a very precise way. For example, they can be always defined with the same number of vertices, and these vertices are always ordered in the same way. The exact knowledge of the way such objects are built allows us to define simplified objects by suppressing or combining specific vertices.

The following example represents the Ishape primitive -12 quad meshes-(see Figure 5). We allow four other levels of details -LOD- for this shape. The order of Ishape vertices is always the same, so we have defined some proper LOD primitives -respectively : 8, 6, 4 quad meshes and one line-(see Figure 6).

Figure 5 - The Ishape primitive

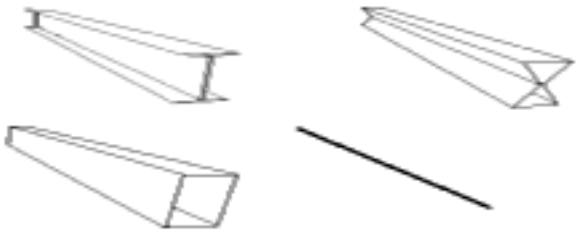


Figure 6 - The four LOD Ishape primitives

Generalized Cylinders An usual way to define complex surfaces is by sweeping a 2D cross-section through a curve in space. The boundary of a cross-section is defined by a series of vertices.

With such methods, it is possible to define shapes such as cylinders or torus parts.

As the number of times a cross-section is repeated and the number of vertices of each cross-section can vary, this gives us an easy way to simplify geometries.

The concept of generalized cylinder can be extended to create half-spheres or spheres simply by adding isolated points at one or the two extremities (Fig. 7). Such primitives can also be easily simplified.

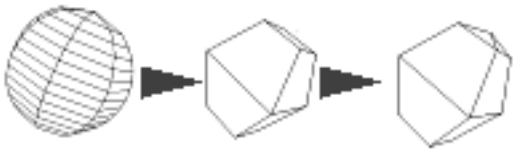


Figure 7 - To simplify a sphere

Our algorithm for simplifying such geometries has the following parameters:

- number of times a cross-section is repeated
- number of vertices of each cross-section
- wished number of times a cross-section is repeated
- wished number of vertices of each cross-section
- ordering of vertices of each cross-section: clockwise or counterclockwise
- two booleans for closing or not the two extremities of the cylinder
- specification of one or two apices to create pyramids at the extremities

Polygonal Models Polygonal models can be simplified by various techniques which have been reviewed in [6].

We mainly use a robust and efficient algorithm which has been described in [9]. The principle is the following: all points which are close to each other are merged to give only one point. The points which are considered as being close to each other are the points which lie inside the cubes of a regular subdivision of the bounding box which surrounds the object to simplify. Although this algorithm is rather brute-force (it does not preserve topology except for simple convex shapes and it is not invariant under translations and rotations), it has the advantage of being easy to implement and fast to excecute. Moreover, it seems to work well for simple convex shapes as is often the case in industrial plants.

Another well-known algorithm, which has the advantage of preserving topology, has been described by Schroeder in [10]. This iterative algorithm suppresses vertices which do not contribute to the salient features of a shape and replace the holes thus created by new triangulations.

Some more elaborate algorithms, such as the one designed by Varshney [13], have the ability to build surfaces wich are constrained to lie whitin some specified epsilon distance from the original surface.

5.2 Culling Methods

Clovis uses a general culling method which computes, possibly on several precessors, the intersection of the bounding boxes of scene objects with the viewing pyramid in order to alleviate the graphics computations.

For our specific purposes, three general methods have been tried to choose which objects to display and with which levels of detail. In an initial stage objects are defined at various levels of detail with the preceding algorithms.

In the first method, all computations are made in real time. We calculate for each object a criterion which allows us to know if this object must be displayed and if so with which level of detail. We use a simple criterion: the distance of the viewer to the object. There is a maximum distance, beyond which all objects are invisible. Despite of its simplicity, this criterion involves too many computations as there are more than 650 objects in the oil refinery. To improve frame rates, computations are not made at every frame, but at constant time intervals and only a portion of the database, some number of randomly chosen objects, is taken into account in these computations.

In the second method, a precomputation phase has been used to split the space into several regions and to compute various levels of detail. The principle of this method is to limit the number of polygons inside each region. After a quota of polygons is chosen for each region, various methods are used to choose the size and position of these regions and also the objects and levels of detail associated with each region. The important thing to notice is the goal we want to achieve in this way: with such algorithms interactivity is independent from the number of initial polygons because the user sees always less polygons that the quota that has been fixed in a first step.

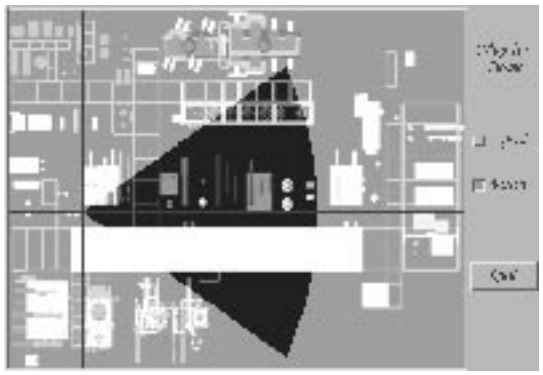
The method works as follows. The ground of the oil refinery is subdivided into several subdomains which contain about the same number of polygons. When the user moves inside one of these subdomains he sees all objects which are inside this region and only some objects in neighboring subdomains. These objects can be displayed with various levels of detail, depending on their size and on their distance to the center of the region. When the user stops, all objects are rendered with full details. In this way, after all regions and their corresponding objects are determined in an initial step, the only computation that is made during visualization consists in determining which region the user is walking in.

The different regions of the ground are arranged either regularly or in a quadtree which is computed in a recursive way. A maximum number of polygons per region is fixed. Then the initial ground which is rectangular is subdivided into four rectangles if it possesses more polygons than this number. Then, all four rectangles are examined successively with the same criterion. If one of them possesses more polygons than wished, it is subdivided into four parts, etc.

In our third and more efficient method, the objects are partitioned into a regular subdivision of the scene into cells and two methods have been tried to know which cells are inside the viewing pyramid.

- Rays are shot from the eyepoint with a Bresenham algorithm to determine which cells are intersected. At some distance, some cells may not be detected and then less little objects are drawn. It is such as a natural selection between little, big objects and distance from the eyepoint.
- In the 2D case, computations may be done in hardware. Cells become pixels and by drawing a triangle into a buffer it is easy to determine which cells are included in this triangle. In general case, we don't have to perform a 3D computations because of the global form of virtual worlds : vertical or horizontal and sometimes the both.

Figure 8 - What do we watch in virtual world ?



(a map program to show and to select the position where we are and what we can see -third method-)

These methods are well adapted to open spaces. More complex visibility precomputations can be done for closed and occluded spaces [16]. Another general method, hierarchical z-buffer visibility [3], consists in embedding the scene in a octree, projecting the bounding box of an octree node and, if this projection visible, examining the subnodes of this node until some objects to display can be found.

6 Some Implementation Issues

6.1 Cleaning of defectuous polygons and generation of normals

One of the specific Clovis features is that it doesn't accept arbitrary polygons. Various constraints must be taken into account. For this reason, in this step, the following tasks are preformed:

- vertices which appear two times in a polygon are removed.
- polygons with less than three vertices are removed.
- non-planar polygons are splitted into triangles.
- vertices which are given in a wrong order are reordered. This problem is due to the way the simplification algorithm has been implemented.
- After the polygonal database is corrected, faces are properly oriented and normals are generated with a recursive algorithm which works as follows. We know which faces are shared by any edge of a polygonal object. An arbitrary face is initially selected and clockwise oriented. All faces that are adjacent to this initial face are then clockwise oriented and so on for the faces which share edges with the faces already examined. At the end of the process, all faces share the same orientation. If these faces constitute a closed surface normals must be directed towards the outside. To know if all faces are correctly oriented, a "pseudo-volume" of the object is computed. If it is negative, the orientation is reversed. It is important to notice that this rather complex algorithm has been made available by SGI as an example of C++ class built to deal with Open Inventor scene graphs. In this way, it was very easy to use it in our Open Inventor program.

  Normals are then intensively used by Clovis either to compute polygonal shading or to cull back-facing polygons. In the initial CAD database, approximately 1/2 of the faces were badly oriented and were thus corrected.

6.2 Clovis Scripts Generation

Clovis is able to interpret programming scripts which are used to set up a virtual world experiment.

As the programming scripts are very specific to the way Clovis works and to the experiments that are described in the next section, we will just give a list of some of the elements that must be known by Clovis in our study:

- the hierarchies of objects used by Clovis: the 3D scene, the virtual counterpart of the Clovis user, the tools manipulated by the virtual user (for example, a crane, its control panel, an arrow to select objects, etc).
- the displacement areas the user is constrained to walk on
- the different areas the universe is split into in order to facilitate walkthroughs
- the methods used to navigate into the scene for immersive or non-immersive exploration
- the objects of the 3D scene that can be manipulated and the way this can be done
- the detection areas of the ladders and staircases and the methods used to climb them

7 Bringing Life to a Virtual Oil Refinery: some significant examples

Specific programs in Clovis code have been used to perform specific tasks. Let us give two examples.

7.1 Navigation

Our main aim is to simulate the walk of a human being inside the oil refinery. In a rather realistic way, his viewpoint is always at the same height from the ground he is walking on, except when he encounters a ladder or a staircase, in which case he can go from a floor to another. To allow such behavior, the following algorithm is used. A detection polygon is located under every ladder. When the user walks on such a polygon, he has then a new degree of freedom: he can go up, he is no longer constrained to move on a plane. He has the choice either to move farther away from the ladder or to climb it up. In the second case, he is made prisoner of the ladder as soon as he has climbed some centimeters. He can then only go up or down until he has reached either the top or the bottom of the ladder. At the top of the ladder he can walk on a limited area which defines a floor of the building. The perimiter of this area has been bounded in order to prevent him from falling down on the ground. He can stay there or use other ladders to continue his walk.

7.2 Interaction

Interaction tasks in a virtual environment can be made in various ways with Clovis. For example, we have simulated the use of a crane. The virtual control panel of this crane can be manipulated to displace a hook which is used to seize objects which can then be moved from one place to another.

Such manipulations can be done during immersive or non-immersive walkthroughs. In the first case, the head of the viewer is tracked by a Polhemus, and is used to update the viewpoint while his hand holds a 3D-joystick which helps him to press the buttons of the control panel.

In non-immersive simulations, viewpoint and objects manipulations are driven either by a 2D-joystick or by the keyboard.

8 Conclusion

This study gives rises to several types of conclusions.

From an application point of view, the combination of various tools has proven to be if great use to give a convenient solution to the complex problem of simulating maintenace operations inside complicated plants. Although some progress remains to be done, the main steps are now clearly defined. Appropritate solutions will certainly be very common in some years.

From a software point of view, some conclusions can be drawn on the use of Open Inventor C++ library and Clovis software package.

Open Inventor is of great interest to deal with CAD data at a high level. Abstract ideas can very rapidly give rise to efficient implementations. But Open Inventor is not very appropriate for high rendering performances. With this respect, Clovis is preferable. It integrates some powerful functions such as parallel computation of the objects that are outside the view in order to cull them. An important feature of Clovis is also its ability to interpret programming scripts and to be connected to external programs via shared memory or TCP−IP protocol.

9 Acknowledgments

First of all, we thank Xavier Lhomme for his active participation to this work at IFP.

We thank also for their numerous contributions the teams of computer graphics experts of the IFP Image Group, the Computer Graphics Center of the Marne-la-Vallée University and Imagis group at Grenoble and in particular V. Bui Tran, J. David, O. Dupuy, P. Vuylsteker, P. Guilloteau and F. Sillion. We thank also N. Tsingos and D. David for their programming skills and S. Beck and D. Pouliquen of Medialab for their reactivity. Finally, we thank the Computer Division of TECHNIP, and in particular M. Avezou, H. Poissonier and F. Haynes for providing us with huge datasets containing millions of polygons that constitute a true challenge for us and our Onyx VTX graphics workstation.

References

[1] Balaguer J. F., Gobbetti E. *i3D: A High-Speed 3D Web Browser*, VRML'95, Nadeau D. R., Moreland J. (Eds), ACM-Press, 1995, pp. 69-76

[2] Funkhouser T.A., Séquin C. H. *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments*, ACM Computer Graphics Proceedings, 1993, pp. 247-254

[3] Green N., Kass M. and Miller G. *Hierarchical Z-buffer visibility*, ACM Computer Graphics Proceedings, 1993, pp. 231-238

[4] Medialab (http://www.demon.co.uk/mlab/clovis.html) *Clovis User's Guide*, 1995

[5] Medialab *Clovis Programmer's Guide*, 1995

[6] Pajon J. L., Collenot Y., Lhomme X., Tsingos N., Sillion F., Guilloteau P., Vuylsteker, Grillon G., David D. *Building and Exploiting Levels of Detail: An Overview and some VRML Experiments*, VRML'95, Nadeau D. R., Moreland J. (Eds), ACM-Press, 1995, pp. 117-122

[7] Pesce P. *VRML: Browsing and Building Cyberspace*, 1995, New Riders

[8] Pichler M., Orasche G., Andrews K., Grossman E., McCahill M. *VRweb: A Multi-Protocol VRML Browser*, VRML'95, Nadeau D. R., Moreland J. (Eds), ACM-Press, 1995, pp. 117-122

[9] Rossignac J. and Borrel P. *Multi-resolution 3D approximations for rendering complex scenes*, Modeling in Computer Graphics, Facidieno B., Kunii T.L. (Eds), Springer-Verlag, 1993, pp. 455-466

[10] Schroeder W.J., Zarge J.A., Lorensen W.E. *Decimation of Triangle Meshes* SIGGRAPH, July 1992, 65-70

[11] SDSC *The SDSC VRML browser: webview* http://www.sdsc.edu/EnablingTech/Visualization/vrml/webview.html

[12] Silicon Graphics *WebSpace* http://www.sgi.com/Products/WebFORCE/WebSpace

[13] Varsney A. *Hierarchical geometric approximation* Doctoral dissertation, University of North Carolina at Chapel Hill, Chapel Hill, NC

[14] Wernecke J. *The Inventor Mentor*, 1994, Addison-Wesley

[15] Wernecke J. *The Inventor Toolmaker*, 1994, Addison-Wesley

[16] Yagel R. and William R. *Visibility Computation for Efficient Walkthough of Complex Environments* Presence: Teleoperators and Virtual Environments, Vol. 2, No. 3, 244-258

Appendix - Funtional Space file format : a simple example

```
SpaceHierarchy {
        World
        Region
}

World Earth {
     gravity
}

Region France {
               world         Earth
               Subdivision   4 4 1
               bboxSize      - - -
               bboxeCenter   - - -
       }
}

ObjectClasses {
       Filter {
           Appearance  wood
           Behaviors {
              Movable {
                 translationAxes axes(Object)
                 rotationAxes axes(Object)
                 initialPosition
                 initialOrientation
                 translationConstraints { free, free,
                                          -getbbox(Object).sizeX/2. , getbbox(Object).sizeX/2.,
                                          -getbbox(Object).sizeY,  free }

                 rotationConstraints    { 10, 70,
                                          free, 14,
                                          free, free  }
              }
           }
       }

      Ladder {
           Appearance  wood
           Behaviors {
              Climbable {
                  language   clovis
                  script     ladder.cod
                  parameters {
                     var1    getbbox(Object)
                     var2    ...
                  }
              }
       }

       Crane {
          Parts part2, part2, part3, part4
          Appearances {
              part1 wood
              part2 steel
              part3
              part4
          }
          Behaviors {
```

```
            part1 Movable {
                translationAxes axes(World)
                rotationAxes axes(World)
            }
            part2 Movable {
                translationAxes axes(part1)
                rotationAxes axes(part1)
                initialPosition   { getbbox(part1).minX + getbbox(part1).sizeX*2./3.,
                                    getbbox(part1).sizeY/2.,
                                    getbbox(part1).maxZ }
                initialOrientation 30, 0, 0
                translationConstraints  FIXED
                rotationConstraintsAroundY { -45, 45 }
            }
            part3 Movable {
                translationAxes axes(part2)
                rotationAxes axes(part2)
                initialPosition   { getbbox(part2).maxX, getbbox(part2).maxY, getbbox(part2).maxZ }
                initialOrientation 0, 40, 0
                translationConstraints  FIXED
                rotationConstraintsAroundY { -45, 45 }
            }
            part4 Movable {
                translationAxes axes(part3)
                rotationAxes axes(part3)
                initialPosition   { getbbox(part3).maxX, getbbox(part3).maxY, getbbox(part3).maxZ }
                initialOrientation 0, 40, 0
                translationConstraints  FIXED
                rotationConstraintsAroundY { -45, 45 }
            }
        }
    }
}


Objects {

        ladder {
            objectclass   Ladder
            region        Unit3
            bboxSize      - - -
            bboxCenter    - - -
            lodCount      -
            pointsCount   ...
            linesCount    ...
            polygonsCount ...
            files         ...
        }

        crane {
            objectclass   Crane
            region        Plant
            parts         object1, object2, object3, object4
        }
}
```