VEGA - A GEOMETRIC MODELLING SYSTEM

by

R.F. Woodbury i G.J. Glass

December, 1983

DRC-U8-OU-33

VEGA

A GEOMETRIC MODELLING SYSTEM

Robert F. Woodbury
Carnegie-Mellon University

Gregory John Glass
Carnegie-Mellon University
10 February 1984

ABSTRACT

VEGA is an interactive geometric modelling system which has been developed at Carnsgie-Meilon University primarily for education in architecture and the arts. Its educational use is twofold: first as a medium for description and manipulation of form to aid in creative work and second as a base package of procedures that may be used by advanced architecture students in the creation of specialized modelling packages.

VEGA is written in PASCAL. There are versions **of VEGA** currently running on VAX 11/780computers under the UNIX and VMS operating systems. **VEGA** has been designed to run on a standalone personal computer. Currently **VEGA** is being converted to **run** on Sun Microsytem and PERQ machines.

VEGA is used in the undergraduate curriculum of the School of Architecture at Carnegie-Mellon University. Its primary use to date has been as a means to introduce geometric modelling to architecture students who have minimal computer experience. VEGA may be v?ewed as a complete geometric modelling package or as a programming aid for development of special purpose geometric modelling programs. To date, one such specialized system, a robot arm design package, has been created as a student project.

The development of the VEGA system is continuing. Current areas of interest include the development of more powerful geometric operations on polyhedra, parametric shapes and assemblies, instancing of sub-assemblies, user definition of primitive shapes and an interactive macro language for the manipulation of form.

1 Introduction

In 1981 the College of Fine Arts at Carnegie-Mellon University initiated a program to bring computers into active use in education in the arts. Computers are seen as a tool which can be directly used in the creative process of the arts. The purpose of what came to be known as the CFA project is to provide an environment for the creation of and instruction in such computer-based tools. The emphasis of the CFA project is on tools and media which can be used in the creative process. Computer-aided instruction is not a goal.

Three principal areas of development within the CFA project have evolved. In the design arts, the initial goal of the prject is the creation of software packages which support three-dimensional modelling, colour rendering and two-dimensional drawing. These packages are seen as being "vanilla flavoured", in that they exist to demonstrate the functionality of a modelling technique rather than attempt to make it directly applicable to a particular discipline. The next phase of the project involves the creation of specialized software systems which directly support specific design disciplines. It is intended that the data types of the "vanilla-flavoured" modellers will be used to structure the special-purpose modellers.

In music, a computer-assisted solfege program is being developed. The immediate purpose of the solfege program is as an assist in the development of cognitive skills of music department majors.

In drama, a Skirpan Autocue system is being used to enrichen the lighting sequences that can be supported in a production.

1.1 The CFA Computing Environment

The computing environment of the College of Fine Arts is distributed spatially about fr.a C c "[1] ^ rt functionally across several different computer systems. There are presently five lab spaces within the College. Three of these directly address the design arts.

The CAD Lab is a joint facility of Architecture and the Computer Science Department of CMU. Its purpose is to provide an environment for research and system development in Computer-Aided Design. The Lab uses a VAX 11/780 running UNIX, an E&S Multi-Picture System running off of a PDP 11/34 and Sun Microsystems.

The CFA Lab is a facility for college wide student use in the design arts. It currently uses Visual500 terminals connected to a VAX 11/780 running VMS and several small workstations based on AED512 terminals and LS111/03 processors. Sun Microsystems are planned for this lab.

The CAD/Design Lab is used by the Department of Design primarily as a research laboratory. An IBM CADAM system is used here.

2 Geometric Modelling

VEGA (Vanilla Environment for Geometric Applications) was developed as part of the CFA project. VEGA is a geometric modeller that supports solid polyhedra related together to form assemblies.

2.1 Definition of an Assembly

It is a rare occurrence in architectural design to be concerned with a single object. Almost every architectural solution is a composite of parts, related together into an assembly.

An assembly is a collection of parts. When viewed from a designer's point of view these parts are complexly interdependent. The conceptual forces that determine a shape are many: location relative to other shapes, size dependent upon geometric constraints, functional requirements(strength, connectivity, mass) machining technology and materials standards to mention a few. Eastman [4] makes a distinction between two types of dependencies; internal and external dependencies. He also states that the recognition of external dependencies is a necessary step to the creation of geometric modelling systems that strongly support the definition of assemblies.

One type of external dependency is relative location. An assembly of parts, for example a precast concrete panel, is composed of many pieces which maintain constant spatial relationships to each other independent of the location of the overall assembly. Some of these pieces may in turn be subassemblies themselves consisting of many pieces.

A modelling system for assemblies should support the groupings of parts into an assembly. An assembly may be defined as being composed of a part or of assemblies of parts. It is no coincidence that this definition of an assembly is similar to the definition of a tree. The data structure which represents an assembly of parts is a tree of relative locations. A part may have a parent, another part to which it is always spatially associated. Similarly a part may have children, which are other parts which are dependent for their spatial location on the location of the parent part.

2.2 Definition of a Part

The part is the basic unit in an assembly. In the real world a part is usually associated with a physical entity which has a boundary between inside space and and outside space. This physical entity is called a solid or polyhedron. Any component of a boundary, for example a face, edge or vertex is located relative to some datum that is used to define the geometry of the boundary. This datum can be thought of as a coordinate system for the boundary of the part. The existance of this coordinate system allows the specification of a boundary independent of the spatial location of the part itself.

A part may have numerous other properties which may be described in non-spatial ways. These properties or attributes of the part do not generally depend explicitly on a description of the boundary of the part.

A part must also have a location in space. For example, it would be meaningless to say that a screw existed without the concept of its existing somewhere. The location of a part is given by specifying a coordinate system somewhere in space at which the part can be defined in its own coordinate system.

These three properties of a part, a spatial boundary, non-geometric attributes and spatial location together define a part.

2.3 The Location Property

The location of any part in an assembly is determined by specifying a transformation relative to some other location. The resulting tree of transformations is called an instance tree or a location graph [4]. In each node of the tree there resides a transformation relative to the parent of the node. For the sake of computational convenience, there is also a transformation which locates the node in global space, or the space defined by the root of the instance tree. The operators on the instance tree automatically maintain data integrity between the two transformations.

Each node in the instance tree may have associated with it two types of properites, a shape and non-geometric attributes. Either of these properties may be a null property. The composite of the three components, an instance or node in the instance tree, a shape and a collection of non-geometric attributes provides enough information for a complete description of some physical part.

A set of operators is used to create, query and modify the instance tree representation. These operators ensure the well-formedness of the instance tree.

2.4 Solids Model

The usefulness of solids modelling for the representation of geometric components is solidly established in CAD. There are many strong reasons to use a solids model. These include:

• The existance of only one data type for all solid geometric artifacts.

© The maintenance of well-formedness conditions.

c The completeness of information necessary for the calculation of all geometric properties of a polyhedron.

A solids model representation may be used as the general representation for the generation of views into any design system.

There are several types of representations for solids models. Requicha [6] a has outlined the following different types: pure primitive instances, spatial enumerations, cell decompositions, simple sweep representations, boundary representations, and constructive solid geometry trees.

Of these, the boundary model, specifically the winged-edge data structure [1] [2] and its variants evolved at Carnegie-Mellon by Eastman et al. [3] has served as the primary representation scheme for solids modelling in our lab. The advantages of the the boundary representation scheme for architectural use include the readiness of display of parts for interactive viewing, the ability to use a large number of primitive shapes without a concommitant large extension of code, and the ability to directly access components of a shape's boundary.

2.4.1 The Winged Edge Data Structure

The winged-edgo data structure represents a solid by defining its boundary as a network of bodies, faces, rings, edges and vertices. [3] [2] Each entity in the data structure carries two types of information; information describing the connectivity of entities to other entities of the same or of different types and information describing the location of the entity in euclidean space. These two types have come to be known respectively as the topology and geometry of a shape. [3]

2.4.2 Euler Operators

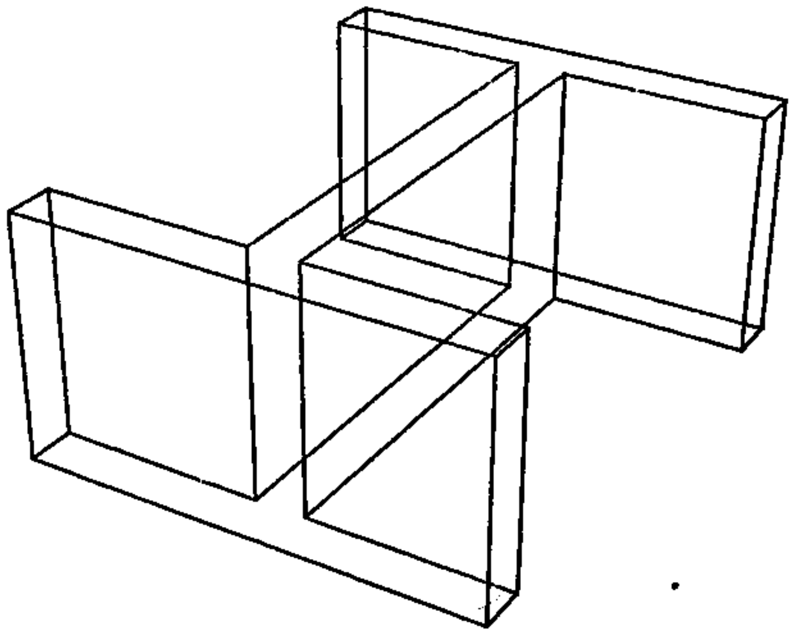
A particular instantiation of the winged edge data structure as a solid or polyhedron is created by use of a set of operators known collectively as the euler operators. These operators guarantee that the resulting graph partially conforms to the well-formedness conditions of non-self-intersection, closure and orientability. [3]

2.4.3 Definition Operators

The euler operators are used to create other higher-level operators which define shapes. These operators create the primitive shapes which the system uses. The primitive shapes currently supported by the modeller are cube, cuboid, cone, frustum, cylinder and extrusion. A new primitive shape can be created by using the euler operators to write a function which returns the required shape. Primitive shapes can be defined parametrically. Figure 1 gives an example of an extruded primitive shape.

Two shapes may be combined to form a resultant. There are four shapes possible from a combination of two shapes. These are the shapes resulting from union, intersection and the two difference operators. Figure 2 gives an example of shape operations on two polyhedra.

Shapes may be modelled. They may be scaled in each of the principal euclidean axes or rotated about the axes. Translation of a shape by changing its local origin is also supported. Figure *3 gives an example of scaling of a shape along one axis.



Figu re 1: Example of an extruded primitive shape

A single shape may be used many times in an assembly of parts. An instance which defines a part may point to a shape which is already "being used" by other instances. When a shape which is used by multiple instances is changed, all of those instances automatically change. This allows a designer to quickly modify all parts which refer to a given shape by modifying only one shape. Since no

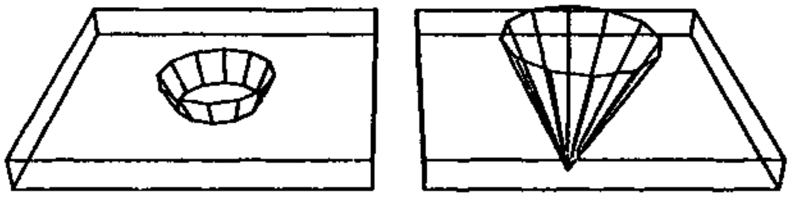
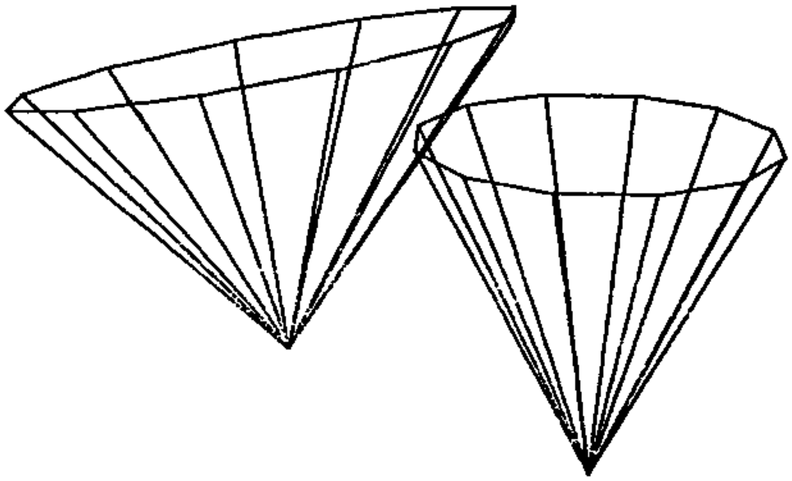


Figure 2: Example of a difference operator on a cone and a cuboid


Figu re 3: Example of a shape scaled along a single axis

canonical form exists for a boundary representation the use of a single shape by many instances provides the only means to maintain a notion of identity between parts. Use of a single shape representation for multiple instances of the same shape also results in a significant reduction of the storage used for a model of given complexity.

2.5 Non-geometric Attributes

A part is not fully described by its location and physical boundary. Other information must be provided to provide complete information. Much of this information can be described textually. Currently, non-geometric information is stored as pairs of text strings referring to the name and value of a part property respectively. Multiple name-value pairs may be used by a single part or instance. In this case, they are grouped together into lists of attributes. An instance may point to one list of attributes. As with shapes multiple instances may point to a single attribute list.

3 The User Interaction Environment

Modeless interaction as proposed by the implementors of Smalltalk80 [7] is the goal that was strived for in the design of the user interaction for this system. This also seemed familiar as we are used to the reverse polish notation of our calculators. This permitted us to perform editing operations on the elements of a design without having to refer to them by name. When the user defines a shape that shape becomes the current shape template and any operations that references a shape will utilize that current shape as an argument. This is similar to the type of interaction provided in the EMACS text editor [5] where the delete word operation deletes the word at the current location. The interaction for a design system can not be exactly the same as that for a text editor because of two problems. The first is that while a text editor manipulates characters that are combined into words that have meaning using a keyboard, there is not a way for a user to "type in" a new shape. We must perform the generation of shapes by either specifying the shape by supplying values that correspond to properties of the shape or by composing primitive shapes together to form new ones. The other problem with the editor model is that it is really working on a one dimensional entity, the string of text being manipulated. In a design system we must manipulate three dimensional elements within a three dimensional space. For doing this task we do not have tools which are as well adopted to this task as the CRT terminal which has been tuned to the task of manipulating characters. ^Jecause of the interactivity of terminal/software systems changes may be easily made on completely unstructured text but the introduction of structure to text (i.e. paragraphs-lines) makes these alterations simpler. It is this imposition of structure that we may use to facilitate alterations in three dimensional design. Because the changes are more complicated during a design process we can not fall back on a well understood structuring as is present in text but users must be able to define their own structural order which includes the definition of the operations that may be used to modify that order.

The interaction was influenced by the interaction of the Smalltalk system and our desire to deal with the problems of working in 3-D space.

The interaction environment presents several types to the user that are not embedded in the underlying model. There are stacks for interaction with instances in the instance tree, for manually input values and strings and for shapes. Notions of current parent in the instance tree and current view are also maintained. A network structure of menus is used to functionally separate different operations and to allow the inclusion of a large number of operators in a limited screen space. An example of a typical menu is shown in Figure 4.

4 Current Uses

VEGA is currently being used in a computer modelling course in the Department of Architecture and for graduate student projects. The purpose of the modelling course is to introduce students to the use of various modelling tools as media for representation in architecture. The course requires a minimal background in computing, particularly an understanding of the concepts of algorithm and data structure. Typical assignments include the demonstration of cubic symmetry, wood framing layout and site planning studies. Students are also asked to prepare a critique of the user interface of the VEGA. Graduate students are using VEGA to write specialized geometric modelling systems. These systems attempt to imbed the semantics of specific disciplines or design activities into a geometric modelling system. The models produced by one special system are data compatible with the VEGA system as a whole. To date, one such project has been completed; a robot arm manipulation package written by one of the authors, Gregory Glass. See figure 5.

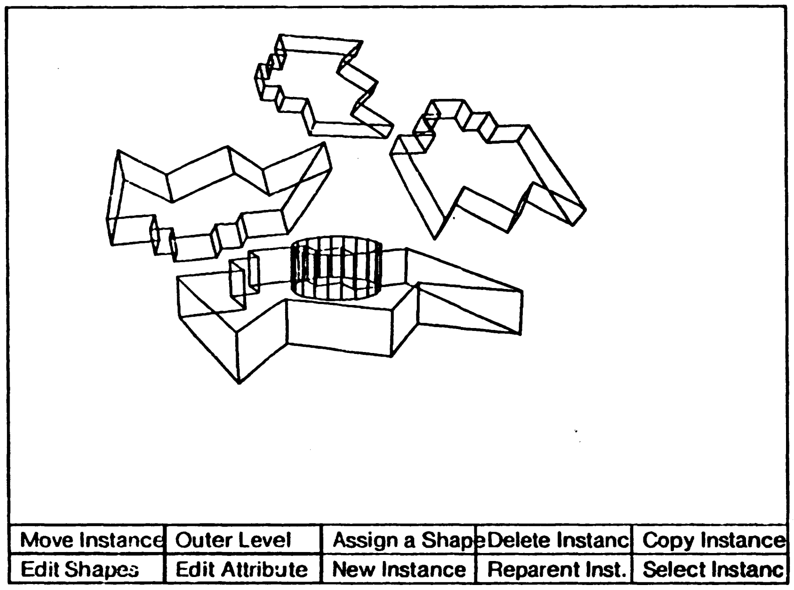


Figure 4: An Example of the Menu and Display

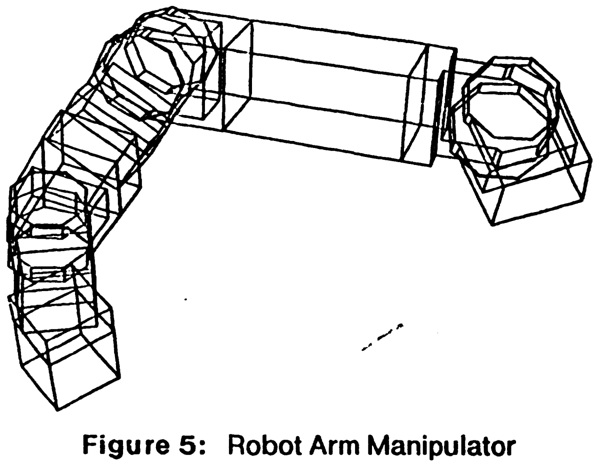


Figure 5: Robot Arm Manipulator

5 Future directions

5.1 Increasing the power of geometric operations on polyhedra

The data structure used for solid polyhedra currently makes little separation between topological and geometric information. This constrains the developer of modelling software to a particular view of the creation of polyhedral form. The development of a data structure for solids that would separate topological and geometric information and allow the definition of polyhedra to be made from either or both in conjunction would significantly increase the shape definition power of the modelling system.

5.2 Polyhedra Definitions

The capabilities of the VEGA to define and alter shapes are now limited to a small number of primitive shapes, some simple modelling operations and the spatial set operators. Much more is possible. We intend to define a group of functions each of which produce a family of shapes. The first of these functions will be the generalized extrusion function, which will be capable of generating simple extrusions, pocket and face extrusions, extrusions with integral holes, shapes of rotation and swept shapes with variable dimension orthogonal to the sweep axis. Other types of special shapes include primitives for the platonic and archimedean solids, for general spatial connectors for linear elements and for parametric shapes. Parametric shapes give the ability to define and manipulate shapes which can be scaled locally. For example, any wide-flange beam can be modelled using the same topological model, which is then scaled appropriately to give a particular instance of a wide-flange beam. Creation of an effective method for the definition and manipulation of parametric shapes involves some research on the representations used to model shapes. Particularly important is the restructuring of the shape data structure to establish a stronger independence between "topological^M and "geometrical" information in the shape model.

For the functions which create these new classes of shapes to be useful, new interaction techniques must be designed and implemented to make the definition and manipulation of these shapes more facile. One function required for true generality of a generalized shape definition function is a general purpose well-formedness test.

5.3 Parametric shapes and assemblies

The definition and modification of assemblies based on parametric value has been addressed previously [4], Often there is a dynamic relation amongst the dimensions of forms that comprise an assembly. An example is a steel truss in a building where the length of the web members depends on the distance between the top and bottom chord. One way to model such a dimensional dependency between parts is separate storage and definition of geometric entities which may be related to polyhedra.

5.4 Limitations on the instance tree concept

The use of an instance tree imposes a strong hierarchical structure onto assemblies. While assemblies are generally constructed in a hierarchical manner, there are many situations which arise in design which do not lend themselves to a hierarchical structure. Research is needed to develop a more general information structuring system for geometric modelling.

5.5 Instancing of subassemblies

An immediatelyImplementable and powerful addition to the modelling system would be the ability to create instances of instances. This would allow us the use of entire subassemblies of parts as if they were a single entity and still retain the notion that these parts are used repetitively in the data structure. The probable response to this issue is the conversion of the current binary tree representation into an acyclic graph of instances. New well-formedness maintainers would have to be written.

5.6 Special Purpose Systems

Two special purpose modeller are in the planning and design stages; a site mapping system that will handle site contours and data overlays, and a system that will prepare the input data files for an energy analysis program.

References

[1] Baumgart, B.G.
Winged edge polyhedron representation.
Stanford Artificial Intelligence Report CS-320, Stanford University, October, 1972.

[2] Braid, I.C.
The Synthesis of Solids Bounded by Many Faces.
Communication of the ACM , April, 1975.

[3] Eastman, Charles M. and Weiler, Kevin.
Geometric Modeling Using the Euler Operators.
Technical Report 78, Institute of Physical Planning, Carnegie-Mellon Univ., February, 1979.

[4] Eastman, Charles M.
The Design Of Assemblies.
Technical Report 11, Institute Of Building Sciences, Carnegie-Mellon Univ., October, 1980.

[5] Meyrowitz, N., van Dam, A.
Interactive Editing Systems: Part 1 and Part 2.
Computing Surveys 14(3):321-415, September, 1982.

[6] Requicha A.A.G., Voelcker H.B.
Advances in Information Systems Science.
Plenum Publishing Corporation, 1981, pages 293-328chapter 5.

[7] Tester, L.
The Smalltalk Environment.
BYTE 6(8):90, August, 1981.