UNDERSTANDING
THE PREPARATION AND ANALYSIS OF
SOLID MODELS

Michael John Muuss

Leader, Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

ABSTRACT

The origins and basic principles of solid modeling will be reviewed. Topics to be covered include a detailed look at the primitive solids and their mathematical descriptions, a review of boolean mathematics for solids combination, and the use of homogeneous coordinates for object positioning. References to various existing solid modeling systems will be given.

As a case study of a modern solid modeling system, the U. S. Army Ballistic Research Laboratory's (BRL) solid modeling system will be described. In addition to an overview of the constructive solid modeling based geometric design editor's (MGED) features and operations, examples of constructing objects using MGED will be presented. MGED's capabilities, command interface, code organization and internal editing states will be discussed. Important considerations in preparing and organizing large models will be presented.

A detailed investigation of the algorithms needed to ray-trace a solids model will be presented. The mathematics of ray-solid intersection will be shown for three primitive solids. Bounding volumes will be discussed, along with boolean expression tree evaluation. The general data structures, code organization, and space partitioning algorithms of BRL's ray tracing package (RT) will be reviewed, and a sample application will be given.

An overview of how ray-tracing of solids models can be applied to compute information about the properties of the structure being modeled will be given. Image rendering, property calculations, and specific engineering assessments will be examined.

13 May 1995

UNDERSTANDING
THE PREPARATION AND ANALYSIS OF
SOLID MODELS

Michael John Muuss

Leader, Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

1. Origins and Principles of Solid Modeling

1.1. Brief history of design

When prehistoric man began fashioning tools and adapting his surroundings to suit him, the history of design began. Driven by his desires, prehistoric man made plans in the form of ideas and mental images. Whether the implementation of these plans was successful or unsuccessful, the plan and goal remained as private mental images, locked within the mind of the individual.

The limitations of this elegant and simple design process were reached when prehistoric man encountered a stone or tree too massive to be dealt with by a single individual. A new dimension was introduced as he attempted to share his ideas and goals with his peers, and organize their aid. Most likely he communicated his ideas by arranging small stones and sticks to form a miniature representation of the initial state of things, and then to manually manipulate his model step by step (no doubt accompanied by grunts, wild gestures, and significant glances) until the desired final configuration was achieved. Then the group went off to implement the plan in full scale. Variations of this design metholodogy persist into the present.

As evolution progressed and the fashionings of man became more and more complex, the necessity for constructing a physical model to communicate ideas grew cumbersome. Detailed models could require substantial effort to construct, and could be difficult to transport. The time required to construct a model might be a substantial fraction of the time to construct the actual object. Furthermore, insight into the construction process could be difficult without disassembling and reassembling the model. Considerations such as these motivated the next level of abstraction: the drawing. The first drawings were probably simple scratchings in earth or sand, depicting the outlines of the objects involved: a two-dimensional wireframe.

With the passage of time, drawings evolved beyond simple engineering representations into recordings of significant events, and art. Drawings were enhanced with color and shading in an attempt to more accurately convey the three dimensional nature of reality. As the field of engineering progressed further, draftsmen began explicitly providing indications of the three dimensional nature of their subjects by representing hidden and false edges as light or broken lines [REQU82], such as in Figure 1.1. Subsequent to this development, drafting advanced very little, and has persisted nearly unchanged up to the present.

The digital computer provides the opportunity for nearly infinite variety in the representation of information within it. Improving the design of complex, expensive objects is always an area ripe for technological improvements, and with the emergence of graphics displays, computer aided design (CAD) packages began to appear. Designers of early CAD packages focused their efforts on the most tedious, time-consuming, and unrewarding aspect of conventional design: the process of converting a designer's sketches and notes into finished engineering drawings suitable for use by a manufacturing facility — the drafting proces. Thus, while computers offered the ability to create arbitrarily powerful mathematical representations for objects, initial CAD systems replicated traditional two-dimensional drafting techniques. Only modest gains in efficiency were made in creating the first version of a design, but tremendous gains were made when design modifications were needed, because the computer retains and redraws the unchanged portions of the drawing.

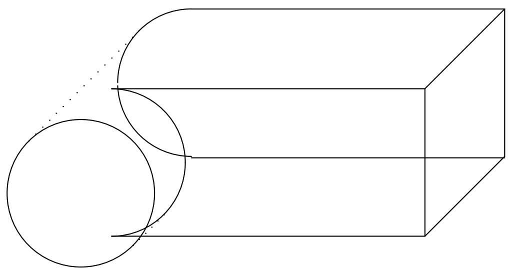

Figure 1.1 — Wireframe Ambiguity

Once a computer aided drafting system has been used to create a computer representation of a design, the designer (and his management) is often tempted to expect more out of the computer system than simply drawings. One would expect that a representation that depicted the boundaries of an object from several views would provide sufficient information for computing a whole variety of useful facts about the model, such as center of mass, volume, cross-sectional area, etc. Two-dimensional drafting systems were poorly prepared for this sort of interrogation. With the addition of appropriate connectivity and material property information, these capabilities became possible, giving rise to 2.5-dimensional drafting systems, which are frequently called 3-dimensional wireframe systems. For limited analysis, 2.5-dimensional models may provide all the required information.

Four major types of difficulties have plagued wireframe systems. First, the user is required to supply a large amount of information, often at a very low level. Because of the drafting heritage, some of this information may be "construction lines" that do not actually contribute to the ultimate shape of any object. Second, because the user is providing such low-level information, objects can easily be defined which can not be physically realized due to non-closed faces and dangling lines. Third, it is possible to construct a wireframe which might represent several different solid objects; additional information is needed to disambiguate the description. Finally, wireframe models may include lines representing false edges such as profile lines and silhouettes, leading to ambiguous interpretations [REQU82]. These issues have driven the refinement of drafting-type systems, but have also focused attention on the quiet evolution of an entirely different approach to representing objects: solid models.

1.2. What is a solid model?

A solid model is a computer description of closed, solid, three-dimensional shapes represented by an analytical framework within which the three-dimensional material can be completely and unambiguously defined [DEIT83]. Solid models differ from drafting-type systems in several important ways: objects are composed of combinations of primitive objects (some quite complex), each of which is complete, unambiguous, physically realizable, and modifiable. Because these properties hold for the primitive objects, they hold for any boolean combinations as well.

Completeness is assured because the representation contains a full description of a piece of solid matter; there is no view-specific information. Because the solid matter is completely described, there is no possibility for ambiguity. For primitive solids defined by specifying parameters to an analytic function, there is no possibility of having missing faces, loose edges, or other similar defects. Systems which offer boundary representations as primitive solids must carefully validate each such solid when it is created. Because solid modeling systems generally do not attempt to compute an exact analytic representation of the surfaces of all the objects, but instead derive the actual surfaces by evaluating the boolean combination expressions in the model interrogation process (such as by ray-tracing), a solid model is always amenable to further modification by boolean combination with other shapes.

These properties guarantee that all the spatial information necessary for any subsequent analysis is directly available from the model representation. Object structure and material properties can be computed at any arbitrary point in the model or along all points of an arbitrary directed ray at any time. Therefore, solid modeling technology is particularly suited to the automation of many manufacturing and analysis tasks.

1.3. The Design Loop

Solid models are very useful for generating drawings or pictures of the modeled object, from any viewpoint. The power of this capability alone usually pays for the cost of developing the model. However, the solid model has a much larger role in the design process than simply automating the production of pictures and engineering drawings. Properly utilized, the solid model becomes the central element in the iterative process of taking a design from idea to prototype design to working design to optimized design. This iterative process is termed the ‘‘design loop’’, and is illustrated in Figure 1.2.

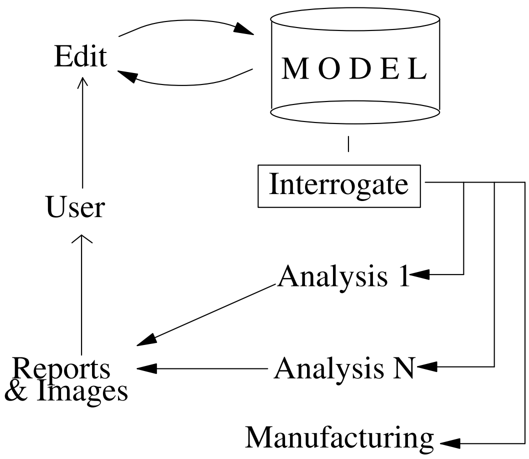

Figure 1.2 — The Design Loop

In a full scale solid modeling system, there is no need for initial drawings; the designer expresses the initial structures directly into the modeling system's editor, just as a modern author creates his ‘‘rough draft’’ directly into a word processor. At the completion of each version of the design, the model is subjected to a battery of analyses appropriate to the function of the object being designed. Strength, volume, weight, level of protection, and other similar evaluations can be reported, along with the production of a variety of images and/or drawings. These automated analyses help identify weaknesses or deficiencies in a design *early in the design process*. By detecting flaws early, the designer has the opportunity to correct his plans before having invested too much time in a bad design, or the designer can switch to an entirely different approach which may seem more promising than the original one. In this way, the solid modeling system allows the designer to concentrate on the important, creative aspects of the design process. Freeing the designer of routine analysis permits designs to be finished in less time than previously required, or allows much more rigorously optimized designs to be delivered in comparable timeframes and at the same cost as unoptimized designs created using older techniques [DEIT85].

A highly interactive modeling system can allow full designs to be completed in a matter of days, where weeks or months may have previously been required [DEIT82]. Furthermore, the modeling system allows sweeping design changes to be made quickly and cheaply, allowing great flexibility in the face of ever changing requirements and markets. The time needed to create a new product can be further decreased by re-utilizing elements of earlier models and then modifying them as appropriate. If such an existing component is entirely suitable for use in a new design, manufacturing and inventory savings should also be realized.

Yet another benefit of developing a solids model is the opportunity to exploit computer aided manufacturing (CAM). In the design evaluation process, numerically controlled (NC) milling machines can be used to produce small quantities of prototype parts with a limited amount of human planning or supervision. When the design is ready for full scale production, optimization of the automatically produced NC programs can result in maximum production rates from factory automation equipment with minimum waste.

In summary, allowing the designer the opportunity to explore and analyze more design options will allow the development of the highest quality product, while also improving the work environment of the designer by eliminating boring, repetitive tasks.

1.4. Model representations

Two major families of solid model representations exist, each with several unique advantages. The first representation, developed by MAGI under contract to BRL [MAGI67] in the mid 1960s, is the combinatorial solid geometry representation (CSG-rep). Solid models of this type are expressed as boolean combinations of primitive solids. Each primitive solid is a geometric entity described by some set of parameters that occupies a fixed volume in space; the choice of primitive solid types available varies from system to system. The simplest solid that can be used is the halfspace [REQU82], defined by the infinite plane $ax + by + cz + d = 0$ plus all points on one side of that plane. Systems which defined all objects in terms of Boolean combinations of halfspaces include SHAPES [LANI73, LANI79] from Draper Laboratories and TIPS-1 [OKIN73a, OKIN78, OKIN73b, OKIN74, OKIN76] from Hokkaido University. While this choice of representation limits these systems to modeling objects with planar faces, and excludes smooth objects (or forces them to be approximated), the simplicity of this representation lends itself to very natural processing by VLSI hardware [KEDE85a, KEDE85b]. Most CSG-rep systems in use today offer quite a variety of primitive solids, ranging from various types of spheres and ellipses, boxes and cones, and solids defined by swept or extruded curves. Some examples of CSG-rep systems which use primitive solids include GDP/GRIN (IBM) [FITZ81], PADL-1 [VOEL74, VOEL78], PADL-2 [BROW82], Series 7000 SMS (Auto-Trol), GMSolid [BOYS82, ROTH80], SynthaVision (MAGI) [GOLD71, GOLD79] (SynthaVision is also marketed as ICEM by CDC, and Solids Modeling II by Applicon), MGED (BRL) [WEAV80, MUUS83], and ProSolid (CAEtec).

The alternative to describing solids with primitives is to adopt a boundary representation (B-rep), of which there are two sub-types, the *explicit* and *implicit* boundary representations. In an explicit boundary representation, each solid is described by an explicit specification of all the points on the surface of the solid, typically by exhaustively listing the vertices of many planar facets. Alternatively, there are implicit boundary representations, where the surface of the solid is described by an analytic function such as Coons patches [COON67], Bezier patches [BEZI74], splines [deBO78], etc. Some examples of B-rep systems (not all of which qualify as solid modelers) are: Alpha_1 (U. Utah) [COHE83], Build-2 (U. Cambridge) [BRAI78, BRAI80, HILY82], CADD (McAuto), CATIA (Dassault/IBM), Compac (Technical U. Berlin) [KRAU76, SPUR76a, SPUR76b, SPUR75, SPUR78], Euclid (Matra Datavision) [BERN75], Euklid (Fides Co.) [EUKLID], Glide (CMU) [EAST77, EAST79], Medusa (Cambridge Interactive Systems, Ltd.), PATRAN (PDA Engineering), Romulus (Shape Data Ltd, Evans&Sutherland) [VEEN79], and Solidesign (Computervision).

1.5. Representation Issues

Boundary representations offer the advantage of being able to naturally model solid objects with arbitrarily shaped surfaces, but can require a large amount of information to achieve acceptable results. Both CSG-reps and B-reps have certain advantages. With pure CSG representation, it can be exceedingly difficult and non-intuitive to attempt to describe sculptured, free-form surfaces. But similarly, implementing operations like boolean intersection and boolean differences *on the fundamental representation itself* can be difficult with pure Boundary representations. Many current B-rep modelers implement boolean operations as an external procedure, because current brute-force schemes to evaluate Boolean operations are not closed. As an example of this, B-spline $\bigcap$ B-spline results in polygons rather than another B-spline. In a boundary representation closed under the set of boolean operations, B-rep $\bigcap$ B-rep $\rightarrow$ B-rep. Thus, pure B-rep systems may be difficult to use for some types of objects, especially those with sculptured surfaces pierced by sharp rectangular gouges [THOM84].

Even though existing systems were listed earlier as being either CSG-rep or B-rep, the reality is that many of these systems are actually hybrids of the two approaches, offering the designer the choice of primitive solids or boundary representations, as appropriate for each task. In practice, the implementation of the CSG-rep and B-rep portions of the software may be quite different, but at the highest level of abstraction each representation is just a different way of viewing the other. Faceted primitives such as boxes and wedges can be thought of as explicit B-reps, and smooth primitives such as spheres and cones can be thought of as implicit B-reps defined by analytic functions. Until one uniform representation can be found, hybrid systems seem both inevitable and desirable.

1.6. Mathematical descriptions of the primitive solids

There are differences between the primitive solids offered by any given modeling system. However, common primitives which are likely to be offered by most systems include the half-space, ellipsoid, truncated general cone, torus, convex polyhedra with small numbers of vertices (typically four to eight), faceted polyhedra, sets of closed B-spline surfaces which taken together define the bounds of a solid object, and curves swept through space. Examples of many of these types of primitives are shown hovering over a mirror in plate 1.1.

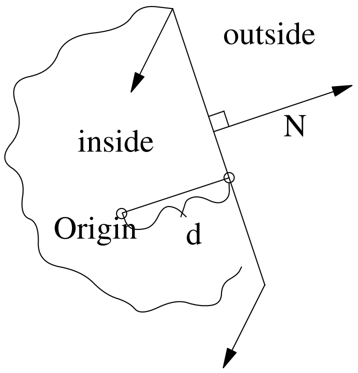

Figure 1.3 — Definition of a Halfspace

1.6.1. Halfspace (HALF)

The simplest solid that can be used is the halfspace, defined by the infinite plane $ax + by + cz + d = 0$ plus all points on one side of that plane. Referring to figure 1.3, another way to define a halfspace is to specify a vector $\vec{\mathbf{N}}$ normal to the surface of the plane with unit length (i.e. $|\vec{\mathbf{N}}| = 1$) and to specify the minimum distance d from the origin to the plane. If $d \geq 0$ then the origin is inside the halfspace, otherwise the origin is outside. More rigorously, any point $\vec{\mathbf{X}}$ is contained within the halfspace if $\vec{\mathbf{N}} \cdot \vec{\mathbf{X}} \leq d$. Points lying on the surface of the plane are considered to be inside the halfspace.

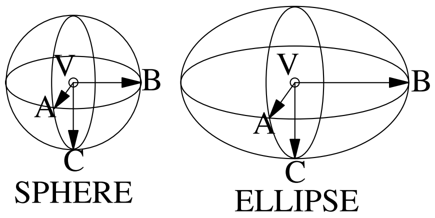

Figure 1.4 — Ellipsoid

1.6.2. Ellipsoid (ELL)

Referring to figure 1.4, an ellipsoid can be defined by specifying a vector $\vec{\mathbf{V}}$ from the origin to the center of the ellipsoid, and by specifying three vectors $\vec{\mathbf{A}}, \vec{\mathbf{B}}, \vec{\mathbf{C}}$ which are mutually perpendicular, and whose magnitudes define the eccentricity of the ellipse. Thus, $|\vec{\mathbf{A}}| = |\vec{\mathbf{B}}| = |\vec{\mathbf{C}}|$ defines a sphere.

1.6.3. Truncated General Cone (TGC)

Referring to figure 1.5, a truncated general cone can be defined by specifying a vector $\vec{\mathbf{V}}$ from the origin to the center of the "lower" ellipse, two perpendicular vectors $\vec{\mathbf{A}}$ and $\vec{\mathbf{B}}$ which define the orientation and

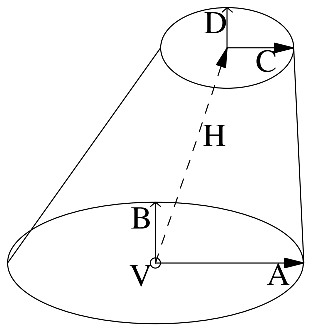

Figure 1.5 — Truncated General Cone

eccentricity of the lower ellipse, a vector $\vec{\mathbf{H}}$ which defines the height and slant of the cone, and two more perpendicular vectors $\vec{\mathbf{C}}$ and $\vec{\mathbf{D}}$ which define the orientation and eccentricity of the upper ellipse.

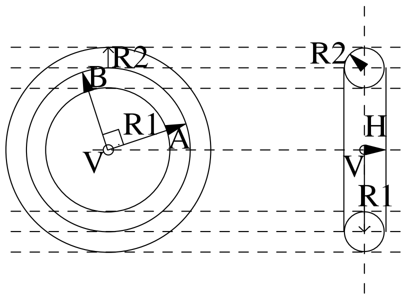

Figure 1.6 — Torus

1.6.4. Torus (TOR)

Referring to figure 1.6, a torus can be defined by specifying a vector $\vec{\mathbf{V}}$ from the origin to the center of the torus, three mutually perpendicular vectors $\vec{\mathbf{A}}$ and $\vec{\mathbf{B}}$ and $\vec{\mathbf{H}}$ which specify the plane containing the torus and the normal to the plane, plus two radii, R_1 which specifies the overall radius of the torus, and R_2 which specifies the radius of the swept circular section of the torus. Some systems may allow the torus to have non-circular shape and cross-section, in which case more parameters would be required.

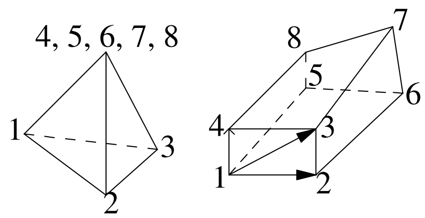

Figure 1.7 — Convex Polyhedron (ARB8)

1.6.5. Convex Polyhedron with 4 to 8 vertices (ARB8)

Referring to figure 1.7, a convex polyhedron can be defined in absolute form by specifying the points which represent the vertices of the faces, or in relative form by specifying a vector $\vec{\mathbf{V}}$ from the origin to the first vertex and seven additional vectors $\vec{\mathbf{R}}_i|_{i=1..7}$ from $\vec{\mathbf{V}}$ to the other vertices. Vectors $\vec{\mathbf{R}}_i$ may coincide or have length=0, $|\vec{\mathbf{R}}_i|_{i=1..7} \geq 0$. Examples of this type of solid are the wedge, box, pyramid, etc.

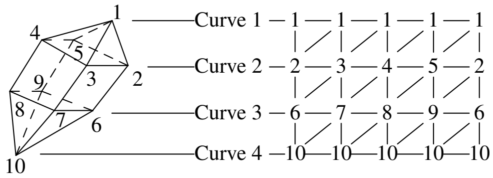

Figure 1.8 — Faceted Polyhedron (ARS)

1.6.6. Faceted Polyhedron (ARS)

Referring to figure 1.8, any faceted polyhedron with logically rectangular structure can be defined by listing N points on each of M curves. In order to make this type of solid more useful, points may overlap, and points are taken in triples to form planar facets. Degenerate facets (lines and points) are discarded.

One can think of this type of polyhedron as representing data obtained by measuring ‘‘waterlines’’, i.e. by placing the object into a basin, adding some water, and measuring N points where the water touches the object. This procedure would be repeated M times to define the whole object. This type of solid is really an explicit boundary representation.

1.6.7. B-spline Solids (SPLINE)

A spline solid is composed of one or more spline surfaces (also called spline patches), which in combination completely enclose some region in space. This type of solid is really an implicit boundary representation. It is important that the model editor ensures that there are no gaps between the spline surfaces so that the primitive is actually solid. An example of one spline surface that might make up a spline solid is shown in plate 1.2.

A clear explanation of the fundamentals of spline surfaces can be had from deBoor [deBO72, deBO78], while a basis for good computer implementations can be found in Riesenfeld [RIES80] and Forrest [FORR78], so they will not be described further. Details of implementations to design sculptured surfaces using the B-spline representation can be found in Cobb [COBB84] and Stay [STAY84].

1.7. The Boolean Math of Combination

Boolean set operations are a natural way to combine shapes. The union operator gives the combined volume of two input shapes, while the intersection operator gives the common volume of two input shapes, and the difference (subtraction) operator acts like a ‘‘cookie cutter’’, giving the volume of the first input shape with the common volume of the two subtracted out. These familiar concepts are illustrated in 4 pairs of plates generated from three primitive solids: a box A, a cylinder B, and a wedge C. The wireframe and shaded image of $A \bigcup B \bigcup C$ are plates 1.3 and 1.4, the wireframe and shaded image of $A - B - C$ are plates 1.5 and 1.6, the wireframe and shaded image of $(A - B) \bigcap C$ are plates 1.7 and 1.8, and the wireframe and shaded image of $C - A - B$ are plates 1.9 and 1.10. Note that the wireframe approximations are for visualization purposes only; while the wireframe outlines are only approximate, the actual solids (e.g., the cylinder) are smooth. For a rigorous treatment of boolean operations in CSG systems, consult Tilove [TILO80].

1.8. Model Structure

Most projects have a natural organization. Similarly, geometric models which are constructed need to have a good, logical structure which is easy to work with and reflects the natural structure of the project. Larger structures should be formed from sub-elements, facilitating top-down design of the overall structure, and bottom-up design of highly detailed subassemblies. This dual strategy allows detail to be added to sub-elements only when needed. Replicated components should not have to be re-described, but instead should be modeled as translated *instances* of the prototype assembly. Finally, there needs to be a mechanism to easily utilize ‘‘model parts bins’’. This allows elements of other models to be incorporated into the current model, either as an exact duplicate, or as a point of departure for further editing.

A directed acyclic graph (DAG) is a good structure to choose to represent this type of structure in a model. A directed acyclic graph is composed of a collection of *nodes* connected by directed *arcs* without

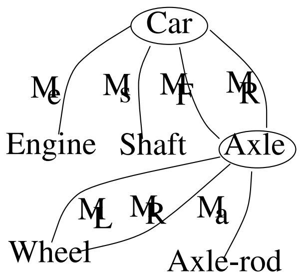

Figure 1.9 — Directed Acyclic Graph for Simple Car

cycles (loops). Each node in the directed acyclic graph has a name. *Leaf nodes* contain a single piece of geometry (a single primitive solid), and *Non-terminal nodes* have one or more departing *arcs*, each *arc* referring to some other node by name. In addition to the referenced name, each *arc* contains a boolean operator and a 4-by-4 transformation matrix. Refer to figure 1.9 for an example of a directed acyclic graph used to express the relationships between various parts of a car.

As the directed acyclic graph is traversed from top to bottom, the total effect is accumulated by multiplying the matrices together. If $\vec{\mathbf{v}}$ is some vector in a solid at a leaf node, and it is transformed by the matrix in a single arc, the transformed vector is

$$\vec{\mathbf{v}}\,' = \vec{\mathbf{v}} * \vec{\mathbf{M}}.$$

When traversing the graph from top to bottom, the matrix on the first arc is considered $\vec{\mathbf{M}}_1$, and the matrix on the final arc is $\vec{\mathbf{M}}_n$. Thus, the final value of a vector is found by

$$\vec{\mathbf{v}}\,' = \vec{\mathbf{v}} * \vec{\mathbf{M}}_n * \cdots * \vec{\mathbf{M}}_1$$

In the example of the car in figure 1.9, a vector in the left wheel on the front axle would be

$$\vec{\mathbf{v}}\,'_{wheel} = \vec{\mathbf{v}}_{wheel} * \vec{\mathbf{M}}_L * \vec{\mathbf{M}}_F.$$

Each matrix is a 4-by-4 Homogeneous Transform Matrix, which may represent any combination of rotation, scaling (uniform or affine), or translation.

$$\vec{\mathbf{v}} = \begin{bmatrix} x & y & z & w \end{bmatrix}$$

$$\vec{\mathbf{v}}' = \begin{bmatrix} x' y' z' w' \end{bmatrix} = \begin{bmatrix} \frac{x'}{w'} & \frac{y'}{w'} & \frac{z'}{w'} & 1 \end{bmatrix}$$

$$\vec{\mathbf{v}}' = \vec{\mathbf{v}} * \vec{\mathbf{M}} = \begin{bmatrix} x & y & z & w \end{bmatrix} * \begin{bmatrix} \alpha r & r & r & 0 \\ r & \beta r & r & 0 \\ r & r & \gamma r & 0 \\ \Delta x & \Delta y & \Delta z & \frac{1}{s} \end{bmatrix}$$

Here, the nine r elements specify the rotation, α, β, and γ specify the single-axis scaling factors (affine transformation), s specifies the uniform scaling factor, and Δx, Δy, and Δz specify the translation distances. Note that the three elements usually used for perspective transformations are zero. Perspective transformations are used for viewing only, and should not be part of a model matrix. For more details on homogeneous coordinates and homogeneous transform matrices, consult pp. 491-501 in Newman and Sproul [NEWM79], and also see pp. 46-88 in Rogers and Adams [ROGE90].

1.9. References

For reviews of the solid modeling field in general, see Requicha [REQU82], Johnson [JOHN85], and Baer [BAER79]. For more information on current solid modelers, the interested reader is referred to Wyvil [WYVI85], Boyse [BOYS82], Muuss [MUUS83], and Brown [BROW82].

1.10. History of Solid Modeling at BRL

The Ballistic Research Laboratory has a long history of innovation in computer science, starting with the world's first electronic digital computer, ENIAC, built for BRL by the Moore School of Engineering. In the early 1960s, BRL retained the Mathematical Applications Group, Inc. (MAGI) to develop a geometric description technique suitable for computer analysis of the survivability of armored military vehicles. This culminated in the development of the MAGIC solid modeling system and shotline (ray-tracing) code in 1966 [MAGI67], which was adopted and documented by a tri-service Joint Technical Coordinating Group for Munitions Effectiveness [JTCG70, JTCG71] and also resulted in a number of other papers being published [GOLD71]. Further development at BRL evolved the MAGIC ray-tracing code into the GIFT (Geometric Information from Targets) ray-tracing code [BAIN75, KUEH79], while MAGI pursued in-house developments which eventually resulted in the commercial Synthavision system [GOLD79, GOLD81].

At BRL, the original GIFT models were prepared laboriously by hand. Each solid was described by dozens of parameters entered on punched cards. In GIFT descriptions, only one level of model structure is available, so good model structure was difficult to achieve. These difficulties notwithstanding, BRL built approximately 150 models for the GIFT system over a span of 14 years. With the exception of simple line drawings of the models, none of these models had ever been seen, and no graphical tools for viewing or manipulating them existed.

In late 1979, the author began an independent, unfunded research effort to investigate the use of three-dimensional graphics display hardware to assist in the development of combinatorial solid geometry descriptions of military vehicles. The success of this effort prompted the development in early 1980 of a special database to describe the hierarchical relationships between the sub-systems of the vehicles being modeled. Geometry so described could be viewed and modified using a special graphics editor called GED [WEAV80], documented in a monochrome movie which demonstrated the capabilities of the GED software.

In 1981, BRL initiated an industry-wide search for computer aided design systems capable of meeting BRL's expanding requirements. Many systems were evaluated, but none met BRL's complex needs, and the decision was made to enhance the in-house software. The result was a state-of-the-art software package capable of meeting BRL's needs for vehicle assessment. In early 1982, the first dedicated DEC PDP-11/70 and associated graphics displays were installed, creating BRL's first production interactive CAD facility [MUUS83]. An early rendering produced by this system is shown in plate 1.11. Later, GED was extended to have multi-display capability; the new version, MGED, went into production in 1984.

In April 1984, a "fast prototype" for a new ray-tracing code was designed and implemented in one week; the team effort to develop the full implementation continued through 1984. In July 1985, the results of recent research in techniques for decreasing the computational expense of ray-tracing [KAPL85] were extended in several simple but significant ways, resulting in substantially higher performance for the RT library. In summer 1985, RT reached production status.

The RT library has permitted other BRL researchers to implement capabilities that were not possible using earlier software, including bi-static lighting models, and models of laser illumination. A project to redesign all of BRL's applications codes around the power of the RT library was begun.

In mid 1985, the RT code was extended for fully parallel operation developed for execution on the BRL HEP Supercomputer. In late 1985, BRL began to experiment with ray-tracing fractal clouds and frequency-controlled random terrain, and began experiments in motion control and animation. Coupling this animation capability with the fully parallel HEP version of the RT code culminated in a 3 minute medium-resolution computer generated animation titled: *"A Quick Run Through The M-2 Bradley"*.

Since the adoption of GED, BRL has been building models of 15 to 20 vehicles annually, usually producing between 3 and 12 variations of each. Most of these models have more than 2000 components, with the larger models often having in excess of 5000 components.

2. BRL's Solid Modeling Editor MGED: A Case Study

There are a significant number of solid modeling systems, each developed for different purposes by different groups. In this paper, details of the design of BRL's solid modeling system [MUUS83] will be presented. This presentation both documents some of the specific capabilities of the BRL system, and also serves as a good case study of the features and functionality that should be present in every solid modeling system. Most features are comparable to those found in commercial systems.

2.1. Philosophy

The role of CAD models at BRL differs somewhat from that of CAD models being built in the automobile and aerospace industries, resulting in some different design choices being made in the BRL CAD software. Because BRL's main use for these models is to conduct detailed performance and survivability analyses of large complex vehicles, it is required that the model of an entire vehicle be completely contained in a single database, suitable for interrogation by the application codes. This places especially heavy demands on the database software. At the same time, a somewhat more modest level of detail is required for these analysis codes than if direct NC machining was the primary goal.

At BRL, there are only a small number of primary designers responsible for the design of a vehicle, and for the construction of the corresponding solid model. Together they decide upon and construct the overall structure of the model, then they perform the work of building substructures in parallel, constantly combining intermediate results into the full model database. Because of the need to produce rapid prototypes (often creating a full design within a few weeks), there is no time for a separate integration stage; subsystem integration must be an ongoing part of the design process.

Once an initial vehicle design is completed, there is usually the need for exploring many alternatives. Typically, between 3 and 12 variations of each design need to be produced, analyzed, and optimized before recommendations for the final design can be made. Also, there is a constantly changing definition of performance; based on new developments it may be necessary to rapidly re-evaluate all the designs of the past several years for trouble spots.

The user interface is designed to be powerful and "expert friendly" rather than foolproof for a novice to use, much like UNIX. However, it only takes about two days for new users to start doing useful design work with MGED. True proficiency comes with a few months practice.

Finally, it is vitally important that the software offer the same capabilities and user interface across a wide variety of display and processor hardware. Government procurement regulations make single-vendor solutions difficult. The best way to combat this is with highly portable software.

2.2. Displays Supported

It is important for a CAD system to have a certain degree of independence from any single display device in order to provide longevity of the software and freedom from a single equipment supplier. The MGED editor supports serial use of multiple displays by way of an object-oriented programmatic interface between the editor proper and the display-specific code. All display-specific code for each type of hardware is thus isolated in a separate *display manager* module. High performance of the display manager was an important design goal. Existing graphics libraries were considered, but no well established standard existed with the necessary performance and 3-dimensional constructs. By having the display manager modules incorporated as a direct part of the MGED editor, the high rates of display update necessary to deliver true interactive response are possible, even when using CPUs of modest power.

An arbitrary number of display managers may be included in a copy of MGED, allowing the user to rapidly and conveniently move his editing session from display to display. This is useful for switching between several displays, each of which may have unique benefits: one might have color capability, and another might have depth cueing. The "release" command closes out MGED's use of the current display, and does an implicit attach to the "null" display manager. This can be useful to allow another user to briefly examine an image on the same display hardware without having to lose the state of the MGED editing session. The "attach" command is used to attach to a new display via it's appropriate display manager routines. If another display is already attached, it is released first. The null display manager also allows the MGED editor to be run from a normal alphanumeric terminal with no graphic display at all. This can be useful when the only tasks at hand involve viewing or changing database structures, or entering or adjusting geometry parameters in numerical form.

Creation of a new display manager module in the "**C**" language [RITC78b] generally takes an experienced programmer from one to three days. The uniform interface to the display manager provides two levels of interactive support. The first level of display support includes the Tektronix 4014, 4016, and compatible displays, including the Teletype 5620 bit-mapped displays. However, while storage-tube style display devices allow MGED to deliver the correct functionality, they lack the rate of screen refresh needed for productive interaction. The second level of support, including real-time interaction, is provided by the Vector General 3300 displays, the Megatek 7250 and 7255 displays, the Raster Technologies Model One/180 display, the Evans and Sutherland PS300 displays with either serial, parallel, or Ethernet attachment, and the Silicon Graphics IRIS 2400 workstation family.

2.3. Portability

Today, the half-life of computer technology is approximately two to three years. To realize proper longevity of the modeling software, it needs to be written in a portable language to allow the software to be moved readily from processor to processor without requiring the modeling software or users to change. Then, when it is desirable to take advantage of the constantly increasing processor capabilities and similarly increasing memory capacity by replacing the installed hardware base, there are a minimum of ancillary costs. Also, it may be desirable to connect together processors from a variety of vendors, with the workload judiciously allocated to the types of hardware that best support the requirements of each particular application program. This distribution of processing when coupled with the fact that users are spread out over multiple locations makes networking a vital ingredient as well.

BRL's strategy for achieving this high level of portability was to target all the software for the UNIX operating system [RITC78a], with all the software written in the "**C**" programming language [KERN78]. All communications software is based upon the TCP/IP protocol suite developed by DARPA [FEIN85] and now formally designated MIL-STD-1777 and MIL-STD-1778. The CAD software is currently running on all UNIX machines at BRL, under several versions of the UNIX operating system, including Berkeley 4.3 BSD UNIX, Berkeley 4.2 BSD UNIX, and AT&T System V UNIX. In addition, there has been a limited subset of the software ported to the Digital Equipment Corporation VAX/VMS operating system, using the VMS C compiler.

The list of manufacturers and models of CPUs that support the UNIX operating system [DEIT84a] is much too lengthy to include here. However, BRL has experience using this software on DEC VAX 11/750, 11/780, 11/785 processors, Gould PN6000 and PN9000 processors, Alliant FX/8 processors (including systems with 8 CPUs), Silicon Graphics IRIS 2400, 2400 Turbo, and 3030 workstations, and the ill-fated Denelcor HEP H-1000 parallel supercomputer.

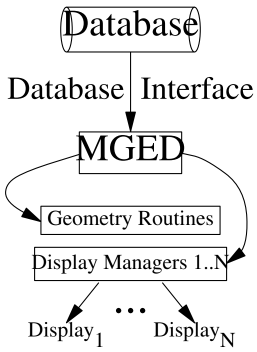

Figure 2.1 — Object-Oriented Design

2.4. Object-Oriented Design

The central editor code has four sets of object-oriented interfaces to various subsystems, including database access, geometry processing, display management, and command parser/human interface, as depicted in figure 2.1. In each case, a common interface has been defined for the set of functions that implement the overall function; multiple instances of these function sets exist when a multiplicity of support exists. The routines in each instance of a function set are completely independent of all the routines in other functions sets, making it easy to add new instances of the function set. A new type of primitive geometry, a new display manager, a new database interface, or a new command processor can each be added simply by writing all the routines to implement a new function set. This approach greatly simplifies software maintenance, and allows different groups to have responsibility for the creation and enhancement of features within each of the function sets.

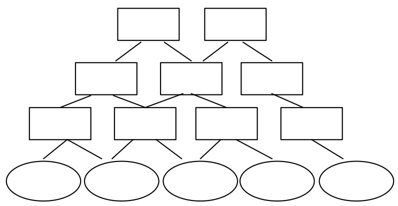

Figure 2.2 — Hierarchy

2.5. Directed Acyclic Graph and Database Details

The database is stored as a single, binary, direct-access UNIX file for efficiency and cohesion, with fixed length records called database *granules*. Each object occupies one or more granules of storage. The user sees and manipulates the directed acyclic graphs like UNIX paths (e.g., car/chassis/door), but in a global namespace. There can be many independent or semi-independent directed acyclic graphs within the same database, each defining different models, as depicted in figure 2.2. The figure also makes heavy use of the *instancing* capability. As mentioned earlier, the *leaves* of the graph are the primitive solids.

Commands exist to import sub-trees from other databases and libraries, and to export sub-trees to other databases. Also, converters exist to dump databases in printable form for non-binary interchange. Within the database, all points are stored as floating point numbers, in millimeters. However, it is important to note that the user edits in his choice of arbitrary working units.

2.6. Interaction Forms

Textual and numeric interaction with the MGED editor is the most precise editing paradigm because it allows exact manipulation of known configurations. This works well when the user is designing the model from an existing drawing, or when all dimensions are known (or are computable) in advance.

The use of a tablet or mouse, knob-box or dial-box, buttons, and a joystick are all simultaneously supported by MGED for analog inputs. Direct graphic interaction via a ‘‘point-push-pull’’ editing paradigm tends to be better for prototyping, developing arbitrary geometry, and fitting together poorly specified configurations. Having both types of interaction capability available at all times allows the user to select the style of interaction that best meets his immediate requirements.

2.7. The Faceplate

When the MGED program has a display device attached, it displays a border around the region of the screen being used along with some ancillary status information. Together, this information is termed the editor ‘‘faceplate’’. In the upper left corner of the display is a small enclosed area which is used to display the current editor state; this is discussed further in the Editor States section, below.

Underneath the state display is a zone in which three ‘‘pop-up’’ menus may appear. The top menu is termed the ‘‘button menu’’ as it contains menu items which duplicate many of the functions assigned to the button box. Having these frequently used functions available on a pop-up menu can greatly decrease the number of times that the user needs to remove his hand from the pointing device (either mouse or tablet puck) to reach for the buttons. An example of the faceplate and first level menu is shown in plate 2.1. The second menu is used primarily for the various editing states, at which time it contains all the editing operations which are generic across all objects (scaling, rotation, and translation). The third menu contains selections for object-specific editing operations. The choices on these menus are detailed below. It is important to note that while some display hardware that MGED runs on has inherent support for pop-up menus included, MGED does not presently take advantage of that support, preferring to depend on the portable menu system within MGED instead. It is not clear whether the slight increase in functionality that might accrue from using display-specific menu capabilities would offset the slight nuisance of a non-uniform user interface.

Running across the entire bottom of the faceplate is a thin rectangular display area which holds two lines of text. The first line always contains a numeric display of the model-space coordinates of the center of the view and the current size of the viewing cube, both in the currently selected editing units. The first line also contains the current rotation rates. The second line has several uses, depending on editor mode. Normally it displays the formal name of the database that is being edited, but in various editing states this second line will instead contain certain path selection information. When the angle/distance cursor function is activated, the second line will be used to display the current settings of the cursor.

It is important to mention that while the database records all positions in terms of millimeters, all numeric interaction between the user and the editor are in terms of user-selected display units. The user may select from millimeters, centimeters, meters, inches, and feet, and the currently active display units are noted in the first display line.

The concept of the ‘‘viewing cube’’ is an important one. Objects drawn on the screen are clipped in X, Y, and Z, to the size indicated on the first status line. This feature allows extraneous wireframes which are positioned within view in X and Y, but quite far away in the Z direction to not be seen, keeping the display free from irrelevant objects when zooming in very close to a particular part of the model.

2.8. Changing the View

At any time in an editing session, the user may add one or more subtrees to the active model space. If the viewing cube is suitably positioned, the newly added subtrees are drawn on the display. (The ‘‘reset’’ function can always be activated to get the entire active model space into view). The normal mode of operation is for users to work with wireframe displays of the unevaluated primitive solids. These wireframes can be created from the database very rapidly.

On demand, the user can request the calculation of approximate boundary wireframes that account for all of the boolean operations specified along the arcs of the directed acyclic graph in the database. This is a somewhat time consuming process, so it is not used by default, but it is quite reasonable to use whenever the design has reached a plateau. Note that these boundary wireframes are not stored in the database, and are generally used as a visualization aid for the designer. Plate 2.2 shows an engine connecting rod. On the left side is the wireframe of the unevaluated primitives that the part is modeled with, and on the right side is the approximate boundary wireframe that results from evaluating the boolean expressions.

Also, at any time the user can cause any part of the active model space to be dropped from view. This is most useful when joining two complicated subsystems together; the first would be called up into the active model space, manipulated until ready, and then the second subsystem would also be called up as well. When any necessary adjustments had been made (perhaps to eliminate overlaps or to change positioning tolerances), one of the subassemblies could be dropped from view, and editing could proceed.

The position, size, and orientation of the viewing cube can be arbitrarily changed during an editing session. The simplest way to change the view is by selecting one of nine built in preset views, which can be accomplished by a simple keyboard command, or by way of a button press or first level menu selection. The user is given the ability to execute a ‘‘save view’’ button/menu function that attaches the current view to a ‘‘restore view’’ button/menu function. The rate of rotation around each of the X, Y, and Z axes can be selected by knob, joystick, or keyboard command. Because the rotation is specified as a rate, the view will continue to rotate about the view center until the rotation rate is returned to zero. Similarly, the zoom rate (in or out) can be set by keyboard command or by rotating a control dial. Also, displays with three or more mouse buttons have binary (2x) zoom functions assigned to two of the buttons. Finally, it is possible to set a slew rate to translate the view center along X and/or Y in the current viewing space, selectable either by keyboard command or control dial. In VIEW state, the main mouse button translates the view center; the button is defined to cause the indicated point to become the center of the view.

This assignment of zoom and slew functions to the mouse buttons tends to make wandering around in a large model very straightforward. The user uses the binary zoom-out button to get an overall view, then moves the new area for inspection to the center of the view and uses the binary zoom-in button to obtain a ‘‘close up’’ view. Plate 2.3 show such a close up view of the engine connecting rod. Notice how the wireframe is clipped in the Z viewing direction to fit within the viewing cube.

2.9. Model Navigation

In order to assist the user in creating and manipulating a complicated hierarchical model structure, there is a whole family of editor commands for examining and searching the database. In addition, on all keyboard commands, UNIX-style regular-expression pattern matching, such as ‘‘*axle*’’ or ‘‘wheel[abcd]’’, can be used. The simplest editor command (‘‘t’’) prints a table of contents, or directory, of the node names used in the model. If no parameters are specified, all names in the model are printed, otherwise only those specified are printed. The names of solids are printed unadorned, while the names of combination (non-terminal) nodes are printed with a slash (‘‘/’’) appended to them.

If the user is interested in obtaining detailed information about the contents of a node, the list (‘‘l’’) command will provide it. For combination (non-terminal) nodes, the information about all departing arcs is printed, including the names of the nodes referenced, the boolean expressions being used, and an indication of any translations and rotations being applied. For leaf nodes, the primitive solid-specific ‘‘describe yourself’’ function is invoked, which provides a formatted display of the parameters of that solid.

The ‘‘tops’’ command is used to find the names of all nodes which are not referenced by any non-terminal nodes; such nodes are either unattached leaf nodes, or tree tops. To help visualize the tree structure of the database, the ‘‘tree’’ command exists to print an approximate representation of the database subtree below the named nodes. The ‘‘find’’ command can be used to find the names of all non-terminal nodes which reference the indicated node name(s). This can be very helpful when trying to decide how to modify an existing model. A related command (‘‘paths’’) finds the full tree path specifications which contain a specified graph fragment, such as ‘‘car/axle/wheel’’. In addition to these commands, several more commands exist to support specialized types of searching through the model database.

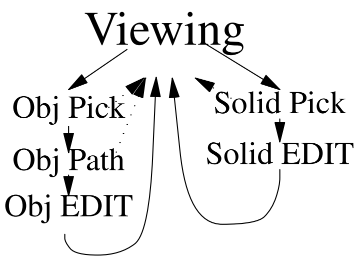

Figure 2.3 — Editor State Transitions

2.10. Editor States

The MGED editor operates in one of six states, with a state transition diagram as shown in figure 2.3. Either of the two PICK states can be entered by button press, menu selection, or keyboard command. The selection of the desired object can be made either by using *illuminate mode*, or by keyboard entry of the name of the object.

Illuminate mode is arranged such that if there are *n* objects visible on the screen, then the screen is divided into *n* vertical bands. By moving the cursor (via mouse or tablet) up and down through these bands, the user will cause each solid in turn to be highlighted on the screen, with the solid's name displayed in the faceplate. The center mouse button is pressed when the desired solid is located, causing a transition to the next state (Object Path, or Solid Edit).

Illuminate mode offers significant advantages over more conventional pointing methods when the desired object lies in a densely populated region of the screen. In such cases, pointing methods have a high chance of making an incorrect selection. However, in sparsely populated regions of the screen, a pointing paradigm would be more convenient, and future versions of MGED will support this.

2.11. Add Primitive

Another family of commands exists to allow the user to add more actual solids (leaf nodes) to the model database. To obtain a precise duplicate of an existing solid (presumably to be changed by a subsequent editing command), the copy (‘‘cp’’) command can be used. It is important to note that the copy operation is different from creating an *instance* of an existing solid; there are occasions to use both operations. If the precise configuration of the solid desired is not important, the ‘‘make’’ command can be used to create a stock prototype solid of the desired type with the given name, which can then be edited to suit. The ‘‘mirror’’ command makes a duplicate of an existing solid reflected about one of the coordinate axes.

If the actual numeric parameters of a solid are known, then the ‘‘in’’ command can be used. In addition to prompting for the descriptions of the full generic primitive solids, this command also accepts abbreviated input formats. For example, a wedge or an RPP can be entered with a minimum of parameters, even though a database ARB8 is created. Similarly, the parameters for a right circular cylinder can be given, resulting in a truncated general cone (TGC) being stored. This is not a very sophisticated way to build solids, but it receives a surprising amount of use.

A number of commands also exist to create new solids with some higher level description. For example, the ‘‘inside’’ command creates a new solid inside an existing solid, separated from the existing solid by specified tolerances. This is quite useful for creating hollow objects such as fuel tanks. It is possible to create a piece of metal plate with a specified azimuthal orientation and fallback angle, or to create an ARB8 (plate) by specifying three points and a thickness, or to create an ARB8 given one point, an azimuthal orientation, and a fallback angle.

2.12. Edit Primitive

There are two classes of editing operations that can be performed on leaf nodes, the primitive solids. The first class of operations are generic operations which can be applied to any type of solid, and the second class of operations are those operations which are specific to a particular type of solid. Generic operations which can be applied to all primitive solids are all those transformations that can be specified by a 4-by-4 homogeneous transformation matrix; each type of transformation can be performed through several types of user direction. Primitives can be rotated around the center of the viewing cube; this rotation can be specified in degrees via keyboard command, or can be controlled by the rotation of a set of control dials or the motions of a three-axis joystick. Translation of a primitive can be specified in terms of a precise new location via keyboard command, or by adjustment of a set of control dials. The primitive can also be moved to a designated position by pointing and clicking with the mouse or tablet. Uniform and single-axis (affine) scaling of a primitive can be controlled by a numeric scale factor via keyboard command, or through repeated analog scaling by pointing and clicking with the mouse or tablet.

Finally, the numeric parameters which define the actual primitive solid can be edited using the user’s choice of UNIX text editor (specified by the EDITOR environment variable). The ‘‘tedit’’ command forms a printed representation of the solid’s numeric parameters in a temporary file and forks a sub-process to allow the user to edit that file. When the text editor exits, MGED regains control, parses the file back into the internal representation of the primitive, and remains in the SOLID EDIT state to allow further modifications.

Each primitive solid also has a variety of editing operations available that are specific to the definition of that solid. Many of these operations are detailed below, but all fall into one of two classes. First, simple scalar parameters such as a radius can be entered numerically via a single parameter keyboard command (‘‘p’’), or they can be adjusted in analog form by pointing and clicking with the mouse or tablet. Secondly, defining vectors can be made to pass through a specified point, or changed to terminate at a given point. The particular point can be specified by entering three parameters to the ‘‘p’’ command, or by pointing and clicking with the mouse or tablet to ‘‘drag’’ the vector.

2.13. Ellipsoid Parameter Edit

Refer to figure 1.4 for an example of the parameters that control the ellipsoid. Most interesting transformations on an ellipsoid can be accomplished by changing it’s orientation in space using the generic operations such as rotation and translation. The shape (eccentricity) of the ellipsoid is controlled by varying the relative lengths of the A, B, and C vectors; these modifications are controlled by this solid-specific menu:

ELLIPSOID MENU
Scale A
Scale B
Scale C
Scale A,B,C

2.14. Truncated General Cone Parameter Edit

Refer to figure 1.5 for an example of the parameters that control the truncated general cone. It is possible to change the length of the H, A, B, C, or D vectors, resulting in a change in height or eccentricity of the end plates. The overall size of the A,B or C,D end plates can be adjusted, or the size of both can be changed together, leaving only the H vector constant. The H vector can be rotated in space, either with the end plates remaining in the same plane, or with the end plates reorienting to be perpendicular to the H vector as it changes. These functions are selected from this menu:

TGC MENU
scale H
scale A
scale B
scale c
scale d
scale A,B
scale C,D
scale A,B,C,D
rotate H
rotate $A \times B$
move end H (rot)
move end H

2.15. Torus Parameter Edit

Refer to figure 1.6 for an example of the parameters that control the torus. While the internal representation of the torus allows for elliptical cross-sections and an elliptical overall shape, those sorts of tori are rarely needed, and this editing menu only permits changing the two radii of a torus with circular cross-section and circular overall shape.

TORUS MENU
scale r1
scale r2

2.16. ARB8 Parameter Edit

Refer to figure 1.8 for examples of an ARB4 and ARB8. ARBs are defined by their vertices, but the placements of the vertices are not entirely free, as each face must be planar. When an ARB is being edited, the user must specify which *edge* is to be moved. Then the user specifies a point in space through which the line containing the edge must pass; this is done either by numerically specifying the parameters with the ‘‘p’’ command, or by pointing and clicking with the mouse or tablet. When editing an ARB with 8 distinct vertices, here is the solid-specific menu that the user would see:

ARB8 MENU
move edge 12
move edge 23
move edge 34
move edge 14
move edge 15
move edge 26
move edge 56
move edge 67
move edge 78
move edge 58
move edge 37
move edge 48

Plate 2.4 shows the wireframe of a box modeled as an ARB8, just after entering SOLID EDIT state but before any editing has taken place. Plate 2.5 shows the solid after edge ‘‘1-4’’ has been moved to a new location.

As the number of distinct vertices in the ARB decreases, the number of menu choices decreases as well, with more and more edge motion choices turning into point motion choices. Consider the ARB4 as an example of this; an example of one is provided in figure 1.7. When editing an ARB with only 4 distinct vertices, here is the solid-specific menu that the user would see:

ARB4 MENU
move face 123
move face 124
move face 234
move face 134
move point 1
move point 2
move point 3
move point 4

There are several keyboard commands that apply only to ARB solids which are being edited in SOLID EDIT state. Once such command is ‘‘mirface’’, which replaces a designated face of the ARB with a copy of an original face mirrored about the indicated axis. Another such command is ‘‘extrude’’, which projects a designated face a given amount in the indicated direction.

2.17. Creating Non-Terminal Nodes

Non-terminal nodes in the directed acyclic graph stored in the database are also called *combinations*. It is possible to extend the definition of a non-terminal node by adding an instance of an existing node to the non-terminal node with an associated boolean operation of union; this is done by the "i" (instance) command. To start with, such an instance has an identity matrix stored in the arc; the user needs to separately edit the arc to move the instance to some other location. If the non-terminal node being extended does not exist, it is created first.

The instance command provides the simplest way to create a reference to a another node. Instances of a whole list of nodes can be added to a non-terminal node by way of the group "g" command. If instances of a list of nodes with non-union boolean operations is to be added to a non-terminal node, the region "r" command accepts a list of (operation, name) pairs, where the single lower case character "u" indicates union, "—" indicates subtraction, and "+" indicates intersection. The first operation specified is not significant. An example of this command might be:

r non-terminal u node1 — node2 + node3

For historical reasons, there is no explicit grouping possible, occasionally forcing the user to create intermediate non-terminal nodes to allow the realization of the desired boolean formula. It is also important to note that for the same reasons there is an *implicit* grouping between union terms, ie

u n1 — n2 + n3 u n4 — n5

is evaluated as

(n1 — n2 + n3) union (n4 — n5)

rather than

((((n1 — n2) + n3) union n4) — n5)

MGED contains a few high-level operations implemented as built-in commands; the most interesting one of these makes an articulated track for a vehicle given the locations of the drive, idler, and road wheels. It is straightforward to create similar domain-specific built-in capabilities as desired, merely by adding another function to the command processor.

2.18. Edit Non-Terminal

Before being able to enter the OBJECT EDIT state (i.e. edit non-terminal), it is necessary to pass through two intermediate states in which the full path of an object to be edited is specified, and the location of one arc along that path is designated for editing. It is possible to create a transformation matrix to be applied above the root of the tree, affecting everything in the path, or to apply the matrix between any pair of nodes. For example, if the full path /car/chassis/door is specified, the matrix could be applied above the node "car", between "car/chassis", or between "chassis/door".

The transformation matrix to be applied at the designated location can be created by the concatenation of operations, each specified through several types of user direction. Trees can be rotated around the center of the viewing cube; this rotation can be specified in degrees via keyboard command, or can be controlled by the rotation of a set of control dials or motions on a three-axis joystick. Translation of trees can be specified in terms of a precise new location via keyboard command, or by adjusting a set of control dials. Tree translation can also be accomplished by pointing and clicking with the mouse or tablet. Uniform and single-axis (affine) scaling of a tree can be controlled by a numeric scale factor via keyboard command, or by way of repeated analog scaling by pointing and clicking with the mouse or tablet.

2.19. Miscellany

MGED has a substantial number of commands which defy easy categorization; some of these commands will be described in this section. The "analyze" command is used to obtain a simple analysis about a single primitive object, including the plane equations, surface area, azimuth and fallback angles of each face, plus the lengths of each edge and the volume of the enclosed space, both in terms of cubic user units, and also in terms of fluid capacity.

Given that the current view is of the evaluated boundaries of the solids, one can obtain a reasonable estimate of the presented area of the components from the current viewing direction by using the "area" command.

It is possible to view, specify, and text-edit information pertaining to the material type and color of various parts of the model tree. This is an interim capability intended to provide enough material properties information for current rendering and analysis purposes until the design of a full material properties database can be finalized.

In addition to a variety of usual database manipulation and status commands, there are commands to compare the current database for name overlap (conflicts) with another database, as well as commands to import and export subtrees to/from the current database. If name conflicts between the two databases do exist, there are commands to rename an individual node without changing any of the references to it (“mv”), or to rename a node and change all the references to it (“mvall”). Another command which is useful for preparing to move subtrees between databases is the “push” command, which adjusts the transformation matrices from the indicated point down to the leaves of the directed acyclic graph, leaving the higher level arcs with identity matrices.

2.20. Output Features

The “plot” command can store an exact image of the current (non-faceplate) display on the screen, either using the System V standard 2-D monochrome UNIX-Plot (*plot(4)*) format, or the BRL 3-D color extended-UNIX-Plot format. These plots can be sent to a disk file, or “piped” directly to a filter process. This can be useful for making hard copies of the current MGED view for showing to others, using a local pen plotter or laser printer.

An important capability even beyond the ability to generate an evaluated boundary wireframe is the ability of MGED to initiate a quick ray-trace rendering of the current view on any nearby framebuffer! This is implemented by using the MGED “rt” command to fork off an instance of the RT program, and sending the RT program a description of the current view with any user-specified options. This allows the designer to use the power of MGED to select the desired view, and then to quickly verify the geometry and light source placement. A 50 by 50 pixel rendering of the current view can usually be done in less than a minute (on a DEC VAX-780 class processor), and allows for general verification before the designer uses the “saveview” command to submit a batch job for a high resolution ray-trace of the same view.

2.21. Animation

The MGED editor includes a number of features which are useful for developing animation tools and scripts. The full description of the current viewing transformation and eye position can be saved in a file, and such a previously saved view can be read back at any time, immediately changing the editor’s view. In addition, the current viewing transformation and eye position can be appended to a file containing a collection of keyframes. Most importantly, a file full of keyframe information, either raw keyframe positions or smoothly interpolated keyframe sequences, can by “played” in real time using MGED, providing a powerful and fast animation preview capability.

As a separate animation capability intended for developing demonstrations and instructional material relating to the use of the MGED editor, all user interactions with the editor can be recorded in a file, along with an indication of the time elapsed between user actions. This file can be adjusted using a normal text editor to remove any errors, or to eliminate dead time where the user stopped to think. Once created, this session script can be replayed through the editor at any time, either to provide a smooth “canned” demonstration before a live audience, or to create a film or videotape.

2.22. Model Building Philosophy

The power of a full directed acyclic graph structure for representing the organization of the database gives a designer a great deal of flexibility in structuring a model. In order to prevent chaos, most designers at BRL choose to design the overall structure of their model in a top-down manner, selecting meaningful names for the major structures and sub-structures within the model. Actual construction of the details of the model generally proceeds in a bottom-up manner, where each sub-system is fabricated from component primitives.

The first sub-systems to be constructed are the chassis and skin of the vehicle, after which a set of analyses are run to validate the geometry, checking for unintentional gaps in the skin or for solids which overlap. The second stage of model construction is to build the features of the main compartments of the vehicle. If necessary for the analysis codes that will be used, the different types of air compartments within the model also need to be described. The final stage of model construction is to build the internal objects to the desired level of detail. This might include modeling engines, transmissions, radios people, seats, etc. In this stage of modeling, the experienced designer will draw heavily on the parts-bin of model components and on pieces extracted from earlier models, modifying those existing structures to meet his particular requirements.

Throughout the model building process it is important for the model builder to choose part names carefully, as the MGED database currently has a global name space, with individual node names limited to 16 characters. In addition, BRL has defined conventions for naming the elements in the top three levels of database structure so that people will be able to easily navigate within models prepared at different times and by different designers, facilitating the integration of design changes into old models with a minimum of difficulty.

3. Interrogating the Model: Ray-tracing

3.1. What Applications Want and Need

The objective of a given application will to a large degree determine the most ‘‘natural’’ form in which the model might be presented. For example, extracting just the *edges* of the objects in a model would be suitable for a program attempting to construct a wire-frame display of the model. Another family of applications exists which needs to be able to find the intersection of small object paths (eg, photons) with the model. Generally, these alternatives are motivated by the representation of a physical process being simulated, and each alternative is useful for a whole family of applications. By choosing the ray sampling density within the Nyquist limit, these applications are well satisfied by extracting ray/geometry intersection information, the well known ‘‘ray-tracing’’ algorithm. However, a mathematical ray has as its cross section a point, while physical objects have significant cross-sectional area. This lack of cross-sectional area can result in sampling inaccuracies, and has lead some applications developers to try to minimize these difficulties. Applications which simulate particles or small rocks approaching the model might benefit from cylinder/geometry intersection capability, and applications which shine beams of light on the model such as spotlights or even highly collimated light such as laser light might benefit from cone/geometry intersection capabilities [AMAN84, KIRK86]. Applications which are attempting to simulate wave effects might be well expressed in terms of plane/geometry intersection curves, and structural analysis routines would probably prefer to obtain the geometry as a collection of connected hyperpatches.

While very recent research has begun to explore techniques for intersecting cylinders, cones, and planes with geometry [KAJI83], ray-tracing is by far the most well developed approach. Fortunately, most applications can function well with approximate, sampled data about the model, rather than needing a precise representation. Data with statistical validity can be obtained by sampling the model with an adequate number of rays and computing the ray/geometry intersections. This approach is also by far the easiest one to implement, as the one-dimensional nature of a mathematical ray makes the intersection equations relatively straightforward.

Given that one has decided to force all applications to interrogate the model strictly by computing ray/geometry intersections, there are still several implementation strategies that could be adopted. The most traditional approach has been batch-oriented, with the user defining a set of ‘‘viewing angles’’, turning loose a big batch job to compute all the ray intersections, and then post-processing all the ray data into some meaningful form. However, the major drawback of this approach is that the application has no immediate control over ray paths, making another batch run necessary for each level of reflection, etc.

In order to be successful, applications need: (1) interactive control of ray paths, to naturally implement reflection, refraction, and fragmenting into multiple subsidiary rays, and (2) the ability to fire rays in arbitrary directions from arbitrary points. Nearly all non-batch implementations have closely coupled a specific application (typically a model of illumination) with the ray-tracing code, allowing efficient and effective control of the ray paths. The most flexible approach of all is to implement the ray-tracing capability as a general-purpose library, and make the functionality available as needed to any application.

3.2. References

The origins of modern ray-tracing come from work at MAGI under contract to BRL, initiated in the early 1960s. The initial results were reported by MAGI [MAGI67] in 1967. Extensions to the early developments were undertaken by a DoD Joint Technical Coordinating Group effort, resulting in publications in 1970 [JTCG70] and 1971 [JTCG71]. The detailed presentation of the fundamental analysis and implementations of the ray-tracing algorithm found in these two documents forms an excellent and thorough review of the principles of solid modeling.

More recently, interest in ray-tracing developed in the academic community, with Kay's [KAY79] thesis in 1979 being a notable early work. One of the central papers in the ray-tracing literature is the work of Whitted [WHIT80]. Model sampling techniques can be improved to provide substantially more realistic images by using the "Distributed Ray Tracing" strategy [COOK84]. For an excellent, concise discussion of ray-tracing, consult pages 363-381 of Rogers [ROGE85].

3.3. Ray -vs- Geometry

The process of intersecting one ray with the model geometry can be decomposed into two distinct tasks. First, the ray must be intersected with each and every primitive solid in the model. Each successful intersection will result in a set of line segments being returned. Secondly, the boolean combinations expressed in the model's directed acyclic graph must be evaluated over this collection of line segments.

We will examine the details of intersecting a ray with a halfspace, a generalized ARB, and an ellipsoid. These three analyses provide all the information necessary to enable one to intersect a ray with any solid possessing a quadratic surface. For example, a truncated general cone can be dealt with as an infinite cylinder (a quadratic) bounded by two planes. The extension to surfaces of higher order is straightforward, and depends only on having a reliable root-finding algorithm. Information about ray-tracing of spline objects can be found in a recent paper by Sweeny [SWEE86].

Rays begin at a point $\vec{\mathbf{P}}$, and proceed infinitely in a given direction $\vec{\mathbf{D}}$. Any point $\vec{\mathbf{A}}$ on a ray may be expressed as a linear combination of $\vec{\mathbf{P}}$ and $\vec{\mathbf{D}}$:

$$\vec{\mathbf{A}} = \vec{\mathbf{P}} + k * \vec{\mathbf{D}}$$

where valid solutions for k are in the range $[0, \infty)$. $(\mathbf{D}_x, \mathbf{D}_y, \mathbf{D}_z)$ are the *direction cosines* for the ray, being the cosine of the angle between the ray and the appropriate axis. While not necessary, in this analysis it is assumed that $\vec{\mathbf{D}}$ is of unit length, i.e. $|\vec{\mathbf{D}}| = 1$.

3.4. Ray/Halfspace Intersection

Consider a plane with arbitrary orientation. This plane partitions space into two half-spaces. Define $\vec{\mathbf{N}}$ to be an outward-pointing unit-length vector which is normal to the plane. Let point $\vec{\mathbf{A}}$ lie anywhere on the plane. Let the ray in question be defined such that any point $\vec{\mathbf{X}}$ on the ray may be expressed as $\vec{\mathbf{X}} = \vec{\mathbf{P}} + k\vec{\mathbf{D}}$.

Determine if the ray is parallel to the surface of the plane, and if not, determine if the ray is entering or exiting the half-space by comparing the direction of the ray with the direction of the normal $\vec{\mathbf{N}}$:

$$\vec{\mathbf{N}} \cdot \vec{\mathbf{D}} = \begin{cases} 0 & \text{parallel, no solution} \\ > 0 & \text{exiting half-space, segment t} = [0, \text{k}] \\ < 0 & \text{entering half-space, segment t} = [\text{k}, \infty) \end{cases}$$

If a solution exists, i.e. $\vec{\mathbf{N}} \cdot \vec{\mathbf{D}} \neq 0$, find parameter k for intersection of ray and plane. To accomplish this, a useful form for the equation of the plane is $\vec{\mathbf{N}} \cdot (\vec{\mathbf{X}} - \vec{\mathbf{A}}) = 0$, where the vector from any point on the plane ($\vec{\mathbf{A}}$) is tested for perpendicularity with the normal $\vec{\mathbf{N}}$. To find k, we substitute the parametric version of $\vec{\mathbf{X}}$, giving:

$$\vec{\mathbf{N}} \cdot (\vec{\mathbf{P}} + k * \vec{\mathbf{D}} - \vec{\mathbf{A}}) = 0$$

$$\vec{\mathbf{N}} \cdot \vec{\mathbf{P}} + k(\vec{\mathbf{N}} \cdot \vec{\mathbf{D}}) - \vec{\mathbf{N}} \cdot \vec{\mathbf{A}} = 0$$

$$k = \frac{\vec{\mathbf{N}} \cdot \vec{\mathbf{A}} - \vec{\mathbf{N}} \cdot \vec{\mathbf{P}}}{\vec{\mathbf{N}} \cdot \vec{\mathbf{D}}}$$

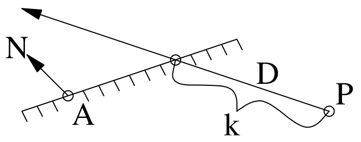

Figure 3.1 — Ray/Halfspace Intersection

3.5. Ray/ARB Intersection

An ARB is a convex volume bounded by 4 (pyramid), 5 (wedge), or 6 (box) planes. This analysis depends on the properties of objects with convex hulls. Let the ray in question be defined such that any point $\vec{\mathbf{X}}$ on the ray may be expressed as $\vec{\mathbf{X}} = \vec{\mathbf{P}} + k\vec{\mathbf{D}}$. Intersect the ray with each of the planes bounding the ARB as discussed above, and record the values of the parametric distance k along the ray. With outward pointing normal vectors, note that the ray *enters* the half-space defined by a plane when $\vec{\mathbf{D}} \cdot \vec{\mathbf{N}} < 0$, is *parallel* to the plane when $\vec{\mathbf{D}} \cdot \vec{\mathbf{N}} = 0$, and *exits* otherwise. Find the *entry* point farthest away from the starting point $\vec{\mathbf{P}}$, i.e. it has the largest value of k among the entry points. The ray enters the solid at this point. Similarly, find the *exit* point closest to point $\vec{\mathbf{P}}$, i.e. it has the smallest value of k among the exit points. The ray exits the solid here.

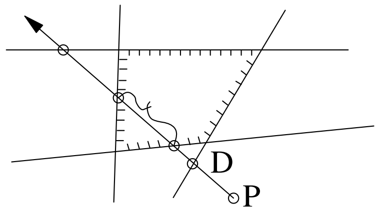

Figure 3.2 — Ray/ARB Intersection

3.6. Ray/Ellipse Intersection

Let the point W belong to the set of points on the surface of an ellipsoid which is defined by $\vec{\mathbf{V}}, \vec{\mathbf{A}}, \vec{\mathbf{B}}, \vec{\mathbf{C}}$, where $\vec{\mathbf{V}}$ is the center of the ellipsoid, and $\vec{\mathbf{A}}$, $\vec{\mathbf{B}}$, and $\vec{\mathbf{C}}$ are mutually perpendicular and define the orientation and shape of the ellipsoid. By affine transformations, map the points on the original ellipsoid into the set of points $\hat{W}$ in the "hat" space which lie on the surface of a unit sphere, located at the origin. For any point X, the corresponding point in the "hat" space is $\hat{X} = S(R(X - V))$, where

$$R(X) = \begin{bmatrix} \vec{A}/(|\vec{A}|) \\ \vec{B}/(|\vec{B}|) \\ \vec{C}/(|\vec{C}|) \end{bmatrix} * X$$

and

$$S(X) = \begin{bmatrix} 1/|\vec{A}| & 0 & 0 \\ 0 & 1/|\vec{B}| & 0 \\ 0 & 0 & 1/|\vec{C}| \end{bmatrix} * X$$

Intersect this ellipse with a ray; points on the ray L are of the form $P + k * D$. This can be be expressed as the intersection of the ray in "hat" space with the unit sphere, where points on the ray $\hat{L}$ are of the form $\hat{P} + \hat{k} * \hat{D}$. The mapping between regular and "hat" space for vectors is:

$$\hat{D} = S(R(D))$$

and for points is:

$$\hat{P} = S(R(P - V))$$

With W defined to be the point where ray L intersects the ellipsoid, and $\hat{W}$ is the point where $\hat{L}$ intersects the unit sphere, then

$$\hat{W} = \hat{P} + \hat{k} * \hat{D}$$

and

$$\hat{W} = S(R(W - V))$$

In order to find the point $\hat{W}$, we refer to the definition for a sphere with unit radius located at the origin

$$x^2 + y^2 + z^2 = 1$$

and substitute in the parametric definition of $\hat{W}$, giving

$$(\hat{P}_x + \hat{k} * \hat{D}_x)^2 + (\hat{P}_y + \hat{k} * \hat{D}_y)^2 + (\hat{P}_z + \hat{k} * \hat{D}_z)^2 = 1$$

Regrouping terms as a polynomial in $\hat{k}$ gives

$$(\hat{D}_x^2 + \hat{D}_y^2 + \hat{D}_z^2)\hat{k}^2 + 2 * (\hat{D}_x \hat{P}_x + \hat{D}_y \hat{P}_y + \hat{D}_z \hat{P}_z)\hat{k} + \hat{P}_x^2 + \hat{P}_y^2 + \hat{P}_z^2 - 1 = 0$$

$$(\hat{D} \cdot \hat{D})\hat{k}^2 + 2(\hat{D} \cdot \hat{P})\hat{k} + [(\hat{P} \cdot \hat{P}) - 1] = 0$$

Solving the quadratic equation for $\hat{k}$ yields 0, 1, or 2 roots

$$\hat{k} = \frac{-2(\hat{D} \cdot \hat{P}) \pm \sqrt{[2(\hat{D} \cdot \hat{P})]^2 - 4(\hat{D} \cdot \hat{D})[(\hat{P} \cdot \hat{P}) - 1]}}{2(\hat{D} \cdot \hat{D})}$$

$$\hat{k} = \frac{-(\hat{D} \cdot \hat{P}) \pm \sqrt{(\hat{D} \cdot \hat{P})^2 - (\hat{D} \cdot \hat{D})[(\hat{P} \cdot \hat{P}) - 1]}}{(\hat{D} \cdot \hat{D})}$$

If a solution for $\hat{k}$ exists, these values of $\hat{k}$ define the intersection point(s) $\hat{W}$. In order to determine W, the relationship between $\hat{k}$ and k needs to be found.

$$\hat{W} = \hat{P} + \hat{k} * \hat{D}$$

$$\hat{W} = S(R(W - V))$$

can be rearranged in terms of W

$$W = V + R^{-1}(S^{-1}(\hat{W}))$$

Then, substituting in the parametric form of $\hat{W}$

$$= V + R^{-1}(S^{-1}[\hat{P} + \hat{k} * \hat{D}])$$

$$= V + R^{-1}(S^{-1}[S(R(P - V)) + \hat{k} * S(R(D))])$$

$$= V + R^{-1}(R(P - V) + \hat{k} * R(D))$$

$$= V + (P - V) + \hat{k} * D$$

$$W = P + \hat{k} * D$$

but,

$$W = P + k * D$$

so $k = \hat{k}$. This useful result is not surprising because affine transformations applied to straight lines yield straight lines, and R and S are both affine transformation matrices. The fact that k is constant for both W and $\hat{W}$ means that the solution for $\hat{k}$ can be determined from $\hat{D}$ and $\hat{P}$ without needing to transform the results out of the ‘‘hat’’ space.

Given that we can find the point $\hat{W}$ where ray $\hat{L}$ intersects the unit sphere and the point W where ray L intersects the ellipsoid, what is the vector normal to the tangent plane at that point? The tangent plane on the unit sphere at $\hat{W}$ has normal vector $\hat{N} = \hat{W}$. (The vector from the origin to any point on the surface of the sphere is pointing in the direction of the tangent at that point).

$$\hat{N} = \hat{W} = S(R(W - V))$$

The tangent plane at $\hat{W}$ transforms back to the tangent plane at W, with normal vector $\vec{N}$:

$$\vec{N} = [(R^{-1} * S^{-1})^T]^{-1}(\hat{N})$$

because if $\vec{H}$ is perpendicular to plane Q, and matrix M maps from Q to $\hat{Q}$, then $[M^T]^{-1}(\vec{H})$ is perpendicular to $\hat{Q}$.

$$\vec{N} = [(S^{-1})^T * (R^{-1})^T]^{-1}(\hat{N})$$

Because $(R^{-1})^T = R$ and $S^T = S$,

$$\vec{N} = [S^{-1} * R]^{-1}(\hat{N})$$

$$= R^{-1} * S(\hat{N})$$

$$= R^{-1}(S(S(R(W - V))))$$

Note that $|\vec{N}| \neq 1$.

3.7. Boolean Evaluation

In the context of the three dimensional model, each ray passes through zero or more objects, but the objects are defined in terms of a boolean combination of primitive solids. The first part of the task is to intersect the ray with each and every primitive solid in the model. For each solid, either the ray will miss, in which case nothing further need be done, or the ray will intersect the solid in one or more line segments. When all of the primitive solids have been intersected, the segments have to be sorted, and ‘‘woven’’ together into intervals. The original ray is broken into a new interval at each point where a primitive solid was entered or exited; the list of primitive solids intersected is constant within each interval. Note that the correct entry and exit normals must be selected each time a segment is broken into an additional interval.

After all the segments are woven together into intervals, each interval needs to be applied to the boolean expression tree. For all primitive solids in the expression tree, if a segment is present within an interval, the expression tree sees a TRUE input, otherwise a FALSE input is presented. For all intervals in which the expression tree returns a TRUE output, the ray has passed through an actual solid object, and that interval is added to the solution list for this ray/model intersection. Note that by keeping the interval list sorted front-to-back in increasing k, and by incrementally weaving new segments into the interval list as they are computed, applications such as lighting models which are only interested in the first intersection with a solid object can be processed much more efficiently by stopping the intersection process after the first hit.

3.8. Bounding Volumes & Space Partitioning

Part of the computational expense of ray-tracing derives from the statement that ‘‘the first part of the task is to intersect the ray with each and every primitive solid in the model’’ results in a an enormous amount of computation. Much of the recent research in ray-tracing techniques has focused on strategies for reducing the the amount of computation required. Overall, the goal is to develop methods to avoid computing the actual ray/solid intersections when the ray does not pass near the solid.

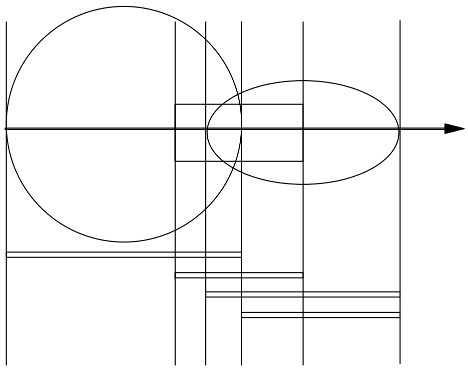

Figure 3.3 — Boolean Evaluation

The first method for eliminating unnecessary ray/solid intersections involves enclosing the solid within a *bounding solid*. Before computing the intersection of a ray with the actual primitive solid, the ray is first intersected with the bounding solid to reject obvious misses. This is only effective if the cost of computing the intersection with the bounding solid is significantly less than the cost of intersecting the ray with the primitive solid. Spheres, axis-aligned right-parallelpipeds (RPPs), and boxes are the three most popularly used bounding volumes. An important issue is the measure of the quality of a bounding volume. The desire is to minimize $VOL_{bound} - VOL_{geom}$, but each bounding volume is a poor fit for some objects. The sphere is a poor bound for any object which is long and thin. Axis-aligned RPPs are poor bounds for any object where the greatest width is not axis aligned. The torus represents a solid that is very hard to bound well at all.

The best bounding sphere has it's center located at the centroid of the primitive solid, with the radius r set to the smallest possible value that encloses the entire primitive solid. Then, to determine if a ray needs to be intersected with the primitive solid, it is merely necessary to determine if the distance from the sphere center $\vec{\mathbf{V}}$ to line $\vec{\mathbf{P}} + k * \vec{\mathbf{D}} \le r$, i.e.

$$\left| (\vec{\mathbf{V}} - \vec{\mathbf{P}}) \times \frac{\vec{\mathbf{D}}}{|\vec{\mathbf{D}}|} \right| \le r$$

When $|\vec{\mathbf{D}}| = 1$, this can be quite fast to compute. Another way to speed the calculation of this would be to compare the magnitude squared to the radius squared, eliminating the necessity for taking a square root in finding the distance.

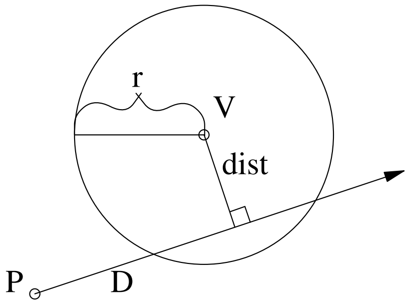

Figure 3.4 — Bounding Sphere

A bounding RPP is defined by six numbers giving the extent of the RPP along each of the three axes, i.e. $MIN_{x,y,z}$ and $MAX_{x,y,z}$. Cyrus and Beck [CYRU77] provide an efficient algorithm for intersecting the ray with the six planes of the RPP. By taking advantage of the fact that all six planes are axis-aligned, the plane intersection equations are greatly simplified, and the following algorithm (written in pseudo-**C**) provides the fastest possible ray/RPP intersection, with a computational cost very close to the cost of computing the distance from a bounding sphere. (Assuming that the division by D_i is replaced with a multiplication by the inverse, which is calculated once per ray).

```
for( i = x, y, z ) {
    if( D_i == 0 ) {
        if( MIN_i > P_i ) return(MISS);
        if( MAX_i < P_i ) return(MISS);
    } else if( D_i > 0 ) {
        /* Heading toward larger numbers */
        if( MAX_i < P_i ) return(MISS);
        k = (MAX_i − P_i)/D_i;
        if( maxK > k ) maxK = k;
        k = (MIN_i − P_i)/D_i;
        if( minK < k ) minK = k;
    } else {
        /* D_i < 0, heading to smaller numbers */
        if( MIN_i > P_i ) return(MISS);
        k = (MIN_i − P_i)/D_i;
        if( maxK > k ) maxK = k;
        k = (MAX_i − P_i)/D_i;
        if( minK < k ) minK = k;
    }
}
if( minK > maxK ) return(MISS);
```

If the ray intersects the RPP at all, it does so in the range (minK, maxK).

Space partitioning is an efficiency strategy which only intersects rays with bounding volumes that are "near" the path of the ray, and avoids finding intersections in irrelevant areas of the model [GLAS84, FUJI85, FUJI86].

Bin-trees enclose all of model space with 6 axis-aligned planes (an RPP), and split the RPP with a single plane carefully chosen to reduce complexity of at least one side of the tree by one solid. Lists of the contents of both new RPPs are made, and then both new RPPs are recursed into until the list of contents is ‘‘small’’. When a ray is to be fired, find the first RPP the ray enters by walking the tree, making binary decisions at each node. When a leaf node is found in the space partitioning tree, fire rays at solids listed (or their bounding volumes).

Oct-Trees operate similarly, except that at each node space is uniformly split by planes in X, Y, and Z.

3.9. RT Library Interface

In order to give all applications interactive control over the ray paths, and to allow the rays to be fired in arbitrary directions from arbitrary points, BRL has implemented its second generation ray-tracing capability as a set of library routines. The RT library exists to allow application programs to intersect rays with model geometry. There are two parts to the interface: preparation routines and the actual ray-tracing routine. Three "preparation" routines exist; the first routine which must be called is dir_build(), which opens the database file, and builds the in-core database table of contents. The second routine to be called is get_tree(), which adds a database sub-tree to the active model space. get_tree() can be called multiple times to load different parts of the database into the active model space. The third routine is rt_prep(), which computes the space partitioning data structures, and does other initialization chores, prior to actual ray-tracing. Calling this routine is optional, as it will be called by shootray() if needed. rt_prep() is provided as a separate routine to facilitate more accurate timing of the preparation and ray-tracing phases of applications.

To compute the intersection of a ray with the geometry in the active model space, the application must call shootray() once for each ray. Ray-path selection for perspective, reflection, refraction, etc, is entirely determined by the applications program, and passed as a parameter to shootray() in the RT ‘‘application’’ structure, which contains five major elements: the vector a_ray.r_pt ($\vec{\mathbf{P}}$) which is the starting point of the ray to be fired, the vector a_ray.r_dir ($\vec{\mathbf{D}}$) which is the unit-length direction vector of the ray, the pointer *a_hit() which is the address of an application-provided routine to call on those rays where some geometry is hit by the ray, the pointer *a_miss() which is the address of an application-provided routine to call on those rays where the ray does not hit any geometry, the flag a_onehit which is set non-zero to stop ray-tracing as soon as the ray has intersected at least one piece of geometry (useful for lighting models), plus various locations for applications to store state (recursion level, colors, etc). Note that the return from the application provided a_hit()/a_miss() routine is the formal return of the function shootray(). The shootray() function is prepared for full recursion so that the application provided a_hit()/a_miss() routines can themselves fire additional rays by calling shootray() recursively before deciding their own return value. In addition, the function shootray() is fully prepared to be operating in parallel with other instances of itself in the same address space, allowing the application to take advantage of parallel hardware capabilities where such exist. This has actually been successfully demonstrated on the Denelcor HEP H-1000 machine and on the Alliant FX/8.

3.10. Sample RT Application

A simple application program that fires one ray at a model and prints the result is included below, to demonstrate the simplicity of the interface to the RT library.

```
struct application ap;
main() {
    dir_build("model.g");
    get_tree("car");
    rt_prep();
    ap.a_point = [ 100, 0, 0 ];
    ap.a_dir = [ -1, 0, 0 ];
    ap.a_hit = &hit_geom;
    ap.a_miss = &miss_geom;
    ap.a_onehit = 1;
    shootray( &ap );
}
hit_geom(app, part)
struct application *app;
struct partition *part;
{
    printf("Hit %s", part->pt_forw->pt_regionp->reg_name);
}
miss_geom(){
    printf("Missed");
}
```

3.11. RT Shootray

To show the simplicity of the overall ray-tracing algorithm, and to provide the reader with enough information to be able to implement a full ray-tracing package, the RT shootray() function is presented.

```
shootray(app)
struct application *app;
{
    if( ray outside model RPP )  return( a_miss() );
    do "push" from box to box  {
        for all solids in box {
            check bounding RPP
            invoke SHOOT on solid, collect segment list
        }
```

```
        bool_weave(): add segs to interval list
        if( app->a_onehit > 0 ) {
            bool_final()
            if( any geometry hit ) break;
        }
    }
    bool_final();
    if( any geometry hit )
        call a_hit();
    else
        call a_miss();
}
```

The overall strategy here is simple, and only the addition of the space partitioning "box" (voxel) notion adds any complexity. If the ray does not even intersect the model RPP, then call a_miss(). Otherwise, examine every box (voxel) that the ray pierces, in increasing distance from the starting point $\vec{\mathbf{P}}$. For every solid inside each voxel, call the SHOOT routine appropriate for that type of solid, and collect the resultant lists of intersection segments. After processing all the solids in the voxel, call bool_weave() to sort the segments into the interval list, creating additional intervals where segments overlap. If the a_onehit flag is set, call bool_final() to evaluate the boolean expressions on the sorted interval list; if any interval evaluates as TRUE, then break the loop and call a_hit(). When all voxels have been examined, call bool_final(), and call a_hit() or a_miss() as appropriate.

It is important to note that the BRL MGED and RT software is available from BRL at no cost by writing to the author. In this way, the author hopes to prevent the implementation of more ray-tracers, and focus attention onto more fruitful areas of research, such as developing more sophisticated algorithms for the analysis of solid models.

4. Analysis of the Model

At any stage of the design, the model can be used to make realistic images and engineering drawings. This capability is so powerful that it ordinarily justifies the construction of the model. However, the real payoff from building a geometric model comes when it is time to analyze it. The model can be subjected to numerous engineering analyses, allowing the effects of varying many parameters to be studied in a controlled way.

4.1. Image Rendering

Many phenomena that are difficult to model with the more traditional scan-line-order rendering of faceted objects can be handled simply and elegantly with ray-tracing, although with a much greater cost in processing time. For example, in most conventional rendering packages, shadows are implemented by way of a subterfuge: computing the *shadow volumes* cast from the light source by each object as a set of "neutral density" polygons, and using them to influence the shading decisions for the actual surfaces underneath them. On the other hand, an illumination model based on ray-tracing merely needs to fire a ray at each light source to determine the total light energy at each point. While the computational requirements of this strategy can be explosive in a model with multiple light sources and many reflective objects, the ease and accuracy of using ray-tracing to model the laws of optics to create very realistic renderings has not yet been surpassed by other techniques.

By dithering the location of the light sources, this can also give a statistically good rendition of shadow penumbra effects. Unlike conventional rendering packages, ray-tracing also makes it easy to deal with objects that are partly or entirely reflective, and with transparent objects that have varying refractive indices. Furthermore, by applying the proper sorts of dither [COOK84], motion-blur, depth-of-field, translucency, and other effects are easily achieved. The ease and flexibility of modeling point-sampled optics with the ray-tracing paradigm is demonstrated by the lighting model code supplied with RT, which implements all of the features listed above (reflection, refraction, perspective views, penumbras, texture maps translucency, etc.) in less than 400 lines of heavily-commented "**C**" code.

An example of these features is shown in plate 4.1. Plate 4.2 shows the outside and inside (skin removed) views of a BMP vehicle, and plate 4.3 depicts an M109 with glass armor. Using simple techniques such as making selected parts of the model invisible or transparent, a designer can create powerful images for communicating the design.

4.2. Animation

Given that lighting model code exists for making static images, it is a relatively straightforward matter to develop code which can animate the position of the "eye" (camera) and light sources within the model, and only a modest additional effort to develop the capability of articulating the model geometry itself. Using animation can be very beneficial when trying to gain comprehension of complex geometry, especially when the object being modeled has never actually been physically built before. There is nothing that communicates complex structures more clearly than observing them passing by smoothly. In addition to assisting the design engineers understanding the geometry that they are developing, an animation capability is also very useful as a sales tool, both for aiding management understanding of the design project, and also to convey a strong message to potential customers.

In addition to being a valuable visualization tool, animation capabilities can be used to experimentally verify that moving assemblies do not interfere with each other, and that adequate clearances exist. This can be extended still further by explicitly modeling the presence of humans as part of the geometry, allowing pre-prototype testing and evaluation of the "human factors" portions of the design. Verification that the design provides adequate space to move around in, easily accessible operating controls, proper provisions for seating, and doorways that are convenient for non-contortionists can be achieved through animation studies.

4.3. Lens Design

The power of the lighting model code can further extended by making a provision to record the paths of all the rays followed when computing the light intensity for each pixel, and recording them in an auxiliary file. In addition to being a fantastic debugging tool, this capability allows one to follow the path of the light rays passing through lenses and being reflected by mirrors while performing image rendering. All these calculations are necessary simply to obtain a rendering of the scene, but if one is actually attempting to use the solid modeling system to study optics, the effects on the actual light ray paths can be obtained with no additional computation.

Studying the paths of light rays as they are repeatedly bent by passing from air to glass and back again has traditionally been a painstaking manual procedure for lens designers. By developing a solid model of some set of optics under consideration, it becomes possible to predict lens behavior, including making a determination of the exact focal length, finding the precise influence of spherical distortions and edge effects, determining the amount of image distortion due to internal reflection and scattering, and finding the level of reflections from the lens mounting hardware. Furthermore, experiments can be conducted to determine the effects of adding or removing baffles, irises, special lens coatings, etc. This is an unusual but highly profitable coupling of solid models and analysis codes.

4.4. Property Calculations

In the design of vehicles, moments and products of inertia play a central role. Particularly when designing aircraft, weights, the center of gravity, and parameters related to inertia are vital aspects of creating a stable design with high performance. For ground vehicles and fixed structures these figures are quite significant, providing a good first estimate of structural loading, and allowing transportation costs to be assessed. Moments of inertia are important in determining what conditions a vehicle may be overturned by maneuvering or impact, and can also help predict the vehicle's handling when maneuvering over rough terrain [DEIT83].

By applying the Fundamental Theorem of integral calculus and performing spatial integration of the geometry as sampled by the ray tracing process, it is possible to compute the presented areas and profile shapes from a given viewpoint, important information to have both for styling and functional design [LEE82]. By combining this spatial integration procedure with a knowledge of the properties of the various materials involved (density, etc), it becomes possible to compute the mass, center of gravity, moments of inertia, overturning moments, and other similar engineering assessments [THOM72]. Plate 4.4 shows one planar set of material property samples from the ray-tracing analysis of an M48 tank. Such material samples comprise the input information for the codes which make the property calculations.

The mass of an object distributed over a region V and having a density $\delta(x, y, z)$ at the point (x, y, z) of V is given by the integral

$$M = \int\int\int \delta(x, y, z)\, dV \ .$$

In the case of sampled geometry, this can be expressed as

$$M = \sum_x \sum_y \sum_z \delta(x, y, z) \ .$$

Similarly, the center of gravity is

$$\left(\frac{\int\int\int x\, \delta(x, y, z)\, dV}{M}, \frac{\int\int\int y\, \delta(x, y, z)\, dV}{M}, \frac{\int\int\int z\, \delta(x, y, z)\, dV}{M} \right),$$

or for sampled geometry

$$\left(\frac{\sum_x \sum_y \sum_z x\, \delta(x, y, z)}{M}, \frac{\sum_x \sum_y \sum_z y\, \delta(x, y, z)}{M}, \frac{\sum_x \sum_y \sum_z z\, \delta(x, y, z)}{M} \right).$$

The moments of inertia I can be found as

$$I = \left(\int\int\int (y^2 + z^2)\, \delta(x, y, z)\, dV, \int\int\int (x^2 + z^2)\, \delta(x, y, z)\, dV, \int\int\int (x^2 + y^2)\, \delta(x, y, z)\, dV \right) .$$

These computational techniques can be extended [BLAS62] to provide the principle second moments and cross products of inertia. Due to the computational complexity of evaluating these mechanical design parameters, they have typically been unavailable to engineers using traditional design approaches.

4.5. Drafting

When a design created on a CAD system is to be produced in traditional manufacturing facilities rather than using a fully automated NC plant, clear and detailed drawings are a necessity. Thus, the production of engineering drawings, including orthogonal views, orthographic projections, and “exploded views” from the solid models is an important, if unglamorous, capability [LAMI86]. Once the model is constructed, the designer may define drawings containing an unlimited number of views, either standard orthogonal views or views from an arbitrary azimuth and elevation. While tolerance specification, fully automatic dimensioning, and tolerance indication is a difficult area [REQU82], recent research is beginning to provide strategies for implementing them [HASH86], and future CAD systems should demand less human intervention than current ones. BRL does not have a well developed capability in this area because of its heavy reliance on color shaded images to communicate final designs.

4.6. Numerical Controlled Milling

Direct numerical control (NC) of a milling machine provides the ultimate in “hard copy” output [VOEL81]. NC milling can be very useful for generating prototypes for initial testing, where the goal is to produce correct toolpath computation in a short amount of elapsed time, but requiring minimal human intervention.

However, NC milling is also very important in the mass production of final versions of the design. In this case, the objective is to compute toolpaths which result in maximum efficiency of the milling machine. A man/machine team effort with human-directed optimization of the automatically produced NC programs can result in maximum production rates from factory automation equipment with minimum waste.

Automatic toolpath generation is still difficult due to the complexity of tool/block interference testing, but various researchers are currently investigating using a ray-tracing paradigm for toolpath planning.

4.7. Structural Analysis

Currently, most structural analysis and hydrodynamics analysis is done using finite element mesh (FEM) techniques. FEM data is normally pre-processed by packages like PATRAN to develop inputs for NASTRAN, EPIC, ANSYS and other similar programs to to actually process the mesh data, including stress testing, computing loading factors, etc.

An automatic interface from the modeling system to a finite element representation is a very important capability. G. Moss at BRL is currently developing a post-processor program which converts CSG-rep solid models into a boundary form suitable for input to PATRAN-G, but this is still an active research effort.

4.8. Vulnerability Assessments

An important extension to simple static and dynamic loading of structures is to consider unexpected dynamic stress caused by a high-energy source impacting the object being designed. This could be as ordinary as a rock striking a car traveling at 80 kilometers per hour, hailstones falling from the sky, or a collision with another vehicle. On the other hand, if the object being designed is intended for use in a military setting, the range of threats which must be considered is greatly enlarged. Not only designers have to worry about airplanes impacting with large birds, hail, and rain, they also need to design airplanes to withstand the buffeting from nearby explosions, flying shrapnel, and various projectiles [WEAV82]. When designing tanks and other ground vehicles, the designer needs to consider the effects of small arms fire, land mines, projectiles, and missiles. Similarly, designers of ocean-going vessels need to concern themselves with the effects of torpedoes, projectiles, and missiles; spacecraft designers need to contend with high-velocity particles, laser beams, and other similar challenges. For all designs, nuclear effects need to be considered.

To conduct a ballistic analysis of point burst effects, BRL has developed a model for lethality and vulnerability estimation which is used to evaluate the effects of antiarmor weapons used against ground vehicles, based upon a ray-tracing approach [BROW81, RING81]. Following perforation of a vehicle armor shell by a penetrator, this code evaluates behind-armor spall cloud effects, described by a cone of debris extending from the point of armor penetration. This process is illustrated in plate 4.5, where the red rods depict the path of the main penetrator, and the blue cone represents the extent of the debris. Soft components located within the spall cone are considered to be destroyed by the debris, while hard components survive unless they are impacted by the main penetrator. This code runs interactively and generates a color display to provide immediate feedback into the design process [OZOL82], an example of which is shown in plate 4.6.

4.9. Nuclear Survivability

When a nuclear weapon is detonated, blast effects, thermal radiation, electro-magnetic pulse, initial nuclear radiation, and fallout all pose threats to equipment and personnel. The initial nuclear radiation has three main components: prompt neutrons, neutron-induced secondary gamma particles, and prompt gamma particles [DEIT83]. The Vehicle Code System (VCS) [RHOD74a, RHOD74b] considers total dose effects by first deterministically calculating the transport of radiation to an envelope that surrounds the vehicle, and then making a stochastic relationship between the radiation at the source envelope and at the detection point inside by using ray-tracing to track each particle through the vehicle. Plate 4.7 shows a rendering of a Tank Test Bed (TTB) concept vehicle, and a version displayed with a false coloring scheme, depicting the proportions of radiation leakage that reached the detection point inside. A surprising result of this test was the unexpectedly high proportion of radiation that entered the vehicle through the narrow front plate; examination of the model did not suggest that it was inadequate. The ability to find and correct this type of deficiency early in the design process is extremely important, and can result in huge cost savings.

4.10. Signatures by "Geometric Sampling"

In the next several sections the topic of predictive signature analysis will be examined for a variety of wavelengths using several slightly different approaches. The development of predictive signature analysis is like a double-edged sword; it can be useful both for developing techniques for reducing the signatures of existing vehicles by guiding product improvement plans, as well as ensuring that new vehicles being designed have an acceptably low signature for frequency ranges of interest, but this kind of analysis can also be used to guide testing and optimize designs for new or improved sensing systems.

It is very important to note that point-sampled geometric optics only bears a statistical relationship to the effects actually obtained from a wave optics analysis (or experiment), and that accounting for phase-related effects is difficult at best. Thus, developing predictive models does not relieve the designer of the responsibility for conducting tests with scale and full size physical models of the prototype, but having these predictive models should make the possibility of unexpected behaviors in the prototypes quite small.

4.11. Vision Assessments

An interesting application for inside-out ray-tracing is to determine the view from within a vehicle through the vision ports or windows. This gives the designers the opportunity to adjust the shape, orientation, and placement of the vision elements of each vehicle so as to optimize the exterior field of view for the vehicle occupants.

Some modern weapons utilize a forward observer to designate a target by laser illumination, with a missile-borne sensor homing in on the reflected beam. This scenario is illustrated in plate 4.8. In order to determine how such a sensor might perform, it is necessary to calculate the laser energy reflected from a variety of candidate targets [DEIT84a]. A related assessment is to determine the amount of optical radiation that might enter a vehicle through each vision port when irradiated with the light from a laser designator. The result of this type of study is usually a cardioid shaped plot showing the energy transmitted as a function of azimuth angle [DEIT83].

4.12. Infrared Modeling

In the design of smart munitions, it is important to know the nature of vehicle signatures over a range of detection bands and for a variety of signal-processing schemes [RAPP83]. Actual predictive infrared modeling is possible based on solid modeling, using finite element techniques. To make predictions about the patterns of heat radiation, it is necessary to calculate a complete internal heat budget for the entire vehicle which accounts for all the sources and radiators of heat, such as engines and cooling fins, and then to take into account external thermal loading due to such factors as the surface of the earth and solar radiation. Heat flow is calculated from node to node in the model, with links between mesh elements characterized by thermal coupling coefficients. The methodology for performing this type of predictive modeling was developed at BRL [RAPP76], but has never been implemented.

4.13. Millimeter Wave Signatures

Dihedral and trihedral metal elements act as particularly efficient reflectors for millimeter wavelength radio transmissions, to the extent that the return from such elements represents the great majority of the return from vehicles [DEIT84b]. BRL has developed a predictive millimeter wave signature model [LACE84] which concerns itself strictly with processing the dihedral and trihedral elements that compose a vehicle. G. Moss at BRL has developed a post-processor program which extracts these features, regardless of size and orientation, from the existing CSG-rep solid models to drive this analysis. An example of the result of this type of processing is shown in Plate 4.9

Because there is a strong correlation between the quantity of dihedral vehicle surface topology and the intensity of the millimeter wave reflection, this feature extraction program can be used by itself without running the full deterministic millimeter wave reflection model to obtain initial estimates of the magnitudes of the millimeter wave signature of a given structure.

4.14. Synthetic Aperture Radar

Synthetic aperture radar (SAR) is a technique with which a variety of image information about a distant object can be obtained by correlating multiple radar samples taken from various positions [TOOM82]. While standard radars only report target backscatter and range information, SAR techniques can resolve distinct scattering regions of a target [DEIT84b]. In 1984, BRL merged the Simulated Radar Image Modeling (SRIM) predictive SAR model from the Environmental Research Institute of Michigan (ERIM) with BRL's raytracing capability. The choice of ray density is based on the frequency of the radar being simulated and the cross-range resolution of the process. Any ray can reflect up to some preset number of times or until it leaves the vicinity of the target. An example of the ray paths for the first and subsequent reflections is shown in plate 4.10. The actual results of the SAR calculation with an M48 vehicle using a single transmit/receive polarization is shown in plate 4.11 These calculations were made in a high-resolution mode unconstrained by practical frequency or coherence considerations of realizable radar systems. The radar signal is propagating from left to right.

4.15. Tradeoff Analysis

The philosophy adopted at BRL has been to develop a broad set of analyses which are supported from the same geometry database [DEIT84b]. These analyses cover the spectrum from engineering decision aids, to design validators, to drafting and milling interfaces, to the generation of manufacturing drawings, to image generation for management comprehension and sales advantage. Key analysis capabilities have been developed to assess the strength, weight, and ''protection'' levels offered by the structures represented by a solid model. Using this analysis information and additional domain-specific applications tools makes it possible to produce highly detailed vehicle designs constructed with a philosophy of *system optimization* right from the start. This should allow the rapid development of vehicles with the desired levels of performance at the best attainable price.

Thanks

The author would like to thank Bob Reschly for his assistance with creating the drawings using PIC and CIP, his supervisors Dr. Stephen Wolff and Dr. Paul Deitz for their unflagging support, Harry Reed for providing an environment that makes good research possible, and Dr. Dave Rogers for persuading me to write it all down.

References

RHOD74a.
W. A. Rhodes et al., *Vehicle Code System (VCS) User's Manual,* Oak Ridge National Laboratory Report ORNL-TM-4648, Oak Ridge TN (August 1974).

AMAN84.
J. Amanatides, "Ray Tracing with Cones," *Computer Graphics (Proceedings of Siggraph '84),* 18, 3 (July 1984).

BAER79.
A. Baer, C. Eastman, and M. Henrion, "Geometric modelling: a survey," *Computer-Aided Design,* 11, 5, pp. 253-272 (September 1979).

BAIN75.
L. W. Bain, Jr. and M. M. Reisinger, *The GIFT Code User Manual; Volume I, Introduction and Input Requirements (U),* BRL Report No. 1802, AD No. B0060371 (July 1975).

BERN75.
Y. J. Bernascon and J. M. Brun, *Automated Aids for the Design of Mechanical Parts (EUCLID),* Tech. Paper MS75-508, Society of Manufacturing Engineers (1975).

BEZI74.
P. E. Bezier, *Mathematical and Practical Possibilities of UNISURF,* Academic Press, New York (1974).

BLAS62.
G. A. Blass, *Theoretical Physics,* p. 102, Appleton-Century-Crofts, New York (1962).

BOYS82.
J. W. Boyse and J. E. Gilchrist, "GMSolid: Interactive Modeling for Design and Analysis of Solids," *IEEE Computer Graphics and Applications,* 2, 2, pp. 27-40 (March 1982).

BRAI78.
I. C. Braid, "New Generations in Geometric Modeling (BUILD)," *Proceedings of Geometric Modelling Project Meeting,* P-78-GM-01, CAM-I, Inc., St. Louis MO (March 14-16, 1978).

BRAI80.

I. C. Braid, R. C. Hillyard, and I. A. Stroud, "Stepwise Construction of Polyhedra in Geometric Modelling" in *Mathematical Methods in Computer Graphics and Design,* ed. K. W. Brodie, pp. 123-141, Academic Press, London (1980).

BROW82.

Brown, "PADL-2: A Technical Summary," *IEEE Computer Graphics and Applications,* 2, 2, pp. 69-84 (March 1982).

BROW81.

F. T. Brown, D. C. Bely, and D. A. Ringers, *The Simple Lethality and Vulnerability Estimator (SLAVE) User's Manual,* BRL Technical Report ARBRL-TR-02282, NTIS AD No. B055277 (January 1981).

COBB84.

E. S. Cobb, *Design of Sculptured Surfaces using the B-spline Representation,* PhD dissertation, University of Utah (June 1984).

COHE83.

E. Cohen, "Mathematical Tools for a Modelers Workbench (Alpha_1)," *IEEE Computer Graphics and Applications* (October 1983).

COOK84.

R. Cook and Porter, L. Carpenter, "Distributed Ray Tracing," *Computer Graphics (Proceedings of Siggraph '84),* 18, 3, pp. 137-145 (July 1984).

COON67.

S. A. Coons, *Surfaces for Computer-Aided Design of Space Forms,* Tech. report MAC-TR-41, Project MAC, MIT, NTIS AD No. 663-504, Cambridge MA (June 1967).

CYRU77.

M. L. Cyrus and J. Beck, *Computer Graphics: Two- and Three-Dimensional Clipping,* Air Force Report AFHLR-TR-77-14, NTIS AD No. A043017 (May 1977).

deBO72.

C. deBoor, "On Calculating with B-splines," *Journal of Approximation Theory,* 6, 1, pp. 50-62 (July 1972).

deBO78.

C. deBoor, *A Practical Guide to Splines,* Applied Mathematical Sciences 27, Springer-Verlag, New York (1978).

DEIT82.

P. H. Deitz, *Solid Modeling at the US Army Ballistic Research Laboratory,* 2, pp. 949-960, Proceedings of the 3rd NCGA Conference (13-16 June 1982).

DEIT83.

P. H. Deitz, *Solid Geometric Modeling - The Key to Improved Materiel Acquisition from Concept to Deployment,* Defense Computer Graphics 83, Washington DC (10-14 October 1983).

DEIT84b.

P. H. Deitz, *Predictive Signature Modeling via Solid Geometry at the BRL,* Sixth KRC Symposium on Ground Vehicle Signatures, Houghton MI (21-22 August 1984).

DEIT84a.

P. H. Deitz, *Modern Computer-Aided Tools for High-Resolution Weapons System Engineering,* DoD Manufacturing Technology Advisory Group MTAG-84 Conference, Seattle WA (25-29 November 1984).

DEIT85.

P. H. Deitz, *The Future of Army Item-Level Modeling,* Army Operations Research Symposium XXIV, Ft. Lee VA (8-10 October 1985).

EAST77.

C. Eastman and M. Henrion, "GLIDE: A Language for Design Information Systems," *Computer*

Graphics (Proceedings of Siggraph '77), 11, 2, pp. 24-33 (July 1977).

EAST79.
C. Eastman and K Weiler, "Geometric Modelling Using the Euler Operators (GLIDE)," *Proc. First Annual Conference on Computer Graphics in CAD/CAM Systems,* pp. 248-259, Cambridge MA (April 9-11 1979).

EUKLID.
M. Engeli, "A language for 30 graphics applications," *International Computer Symposium '73,* North-Holland Press (1974).

EUKLID.
M. Engeli and V. Hrdliczka, *EUKLID: eine Einfuhrung,* Fides Co., Zurich (1974).

FEIN85.
E. J. Feinler, O. J. Jacobsen, M. K. Stahl, and C. A. Ward, *DDN Protocol Handbook, NIC 50004,* DDN Network Information Center, Menlo Park CA (December 1985).

FITZ81.
W. Fitzgerald, F. Gracer, and R. Wolfe, "GRIN: Interactive Graphics for Modeling Solids," *IBM Journal of Research and Development,* 25, 4, pp. 281-294 (July 1981).

FORR78.
A. R. Forrest, "A Unified Approach to Geometric Modelling," *Computer Graphics (Proceedings of Siggraph '78),* 12, 3 (August 1978).

FUJI85.
A. Fujimoto and et al., "Accelerated Ray Tracing," *Proceedings of CG Tokyo '85,* Tokyo (April 1985).

FUJI86.
A. Fujimoto, C. G. Perrott, and K. Iwata, "Environment for Fast Elaboration of Constructive Solid Geometry" in *Advanced Computer Graphics,* ed. T. L. Kunii, pp. 20-33, Springer-Verlag (1986).

GLAS84.
A. S. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications,* 4, 10 (1984).

GOLD71.
R. Goldstein and L. Malin, "3-D Visual Simulation," *Simulation,* 16, 1, pp. 25-31 (1971).

GOLD79.
R. Goldstein and L. Malin, "3D Modelling with the Synthavision System," *CAD/CAM Systems (Proceedings of the First Annual Conference Computer Graphics),* pp. 244-247, Cambridge MA (9-11 April 1979).

GOLD81.
R. Goldstein, "Defining the Bounding Edges of a Synthavision Solid Model," *Proc. 18th Design Automation Conference,* pp. 457-461, Nashville TN (June 29-July 1 1981).

HASH86.
J. Hashimoto, H. Fukushima, and W. Kowaguchi, "High Performance CAD System with Full Automatic Dimensioning and Multi-Modeling Features Based on Engineering Workstation" in *Advanced Computer Graphics,* ed. T. L. Kunii, pp. 358-378, Springer-Verlag (1986).

HILY82.
R. C. Hillyard, "The Build Group of Solid Modelers," *IEEE Computer Graphics and Applications,* 2, 2, pp. 43-52 (March 1982).

JOHN85.
R. Johnson, "Capabilities of Solid Modeling," *Computer Graphics World,* pp. 50-64 (November 1985).

THOM72.
G. B. Thomas Jr., *Calculus and Analytic Geometry,* Addison-Wesley (February 1972).

JTCG70.

JTCG, *MAGIC Computer Simulation, Vol. 1, User Manual,* Joint Technical Coordinating Group for Munitions Effectiveness, 61JTCG/ME-71-7-1 (July 1970).

JTCG71.

JTCG, *MAGIC Computer Simulation, Vol. 2, Analyst Manual,* Joint Technical Coordinating Group for Munitions Effectiveness, 61JTCG/ME-71-7-2-1 (May 1971).

KAJI83.

J. T. Kajiya, "New Techniques for Ray Tracing Procedurally Defined Objects," *Transactions on Graphics,* 2, 3, pp. 161-181 (July 1983).

KAPL85.

M. R. Kaplan, *Space-Tracing, a Constant Time Ray-Tracer,* Siggraph '85 Tutorial ''State of the Art in Image Synthesis'', San Francisco CA (July 22-26, 1985).

KAY79.

D. S. Kay, *Transparency, Refraction, and Ray Tracing for Computer Synthesized Images,* Cornell Univ (Jan 1979).

KEDE85a.

G. Kedem, *Computer Structures and VLSI Design for Curve-Solid Classification,* Siggraph '85 Tutorial ''VLSI for Computer Graphics'', San Francisco CA (July 23, 1985).

KEDE85b.

G. Kedem and J. L. Ellis, *Computer Structures for Curve-Solid Classification in Geometric Modeling,* Siggraph '85 Tutorial ''VLSI for Computer Graphics'', San Francisco CA (July 23, 1985).

KERN78.

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Bell Laboratories, Murray Hill, NJ (1978).

KIRK86.

D. B. Kirk, "The Simulation of Natural Features Using Cone Tracing" in *Advanced Computer Graphics,* ed. T. L. Kunii, pp. 129-144, Springer-Verlag (1986).

KRAU76.

F. L. Krause and Vassilakopoulos, "A way to Computer Supported Systems for Integrated Design and Production (COMPAC)," *IFIPS CAD Working Conference,* Austin TX (February 1976).

KUEH79.

G. G. Kuehl, L. W. Bain, Jr., and M. M. Reisinger, *The GIFT Code User Manual; Volume II, The Output Options (U),* US Army ARRADCOM Technical Report No. ARBRL-TR-02189, AD No. A078364 (September 1979).

LACE84.

J. Lacetera, *Deterministic Modeling of Tank Targets for MMW Radar Systems,* Sixth KRC Symposium on Ground Vehicle Signatures, Houghton MI (21-22 August 1984).

LAMI86.

L. G. Lamit and V. Paige, "The Influence of CADD on Teaching Traditional Descriptive Geometry and Orthographic Projection" in *Advanced Computer Graphics,* ed. T. L. Kunii, pp. 473-484, Springer-Verlag (1986).

LANI73.

J. H. Laning, D. A. Lynde, and V. Moreggia, *SHAPES User's Manual,* Charles Stark Draper Laboratory, Inc., Cambridge MA (May 1973).

LANI79.

J. H. Laning and S. J. Madden, "Capabilities of the SHAPES System for Computer Aided Mechanical Design," *Proc. First Annual Conference on Computer Graphics in CAD/CAM Systems,* pp. 223-231, Cambridge MA (April 9-11, 1979).

LEE82.

Y. T. Lee and A. A. G. Requicha, "Algorithms for Computing the Volume and Other Integral

Properties of Solid Objects," *Communications of the ACM,* 25, 9, pp. 635-650 (September 1982).

MAGI67.

MAGI, *A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles,* MAGI Report 6701, AD847576 (August 1967).

MUUS83.

M. J. Muuss, K. A. Applin, R. J. Suckling, G. S. Moss, E. P. Weaver, and C. Stanley, *GED: An Interactive Solid Modeling System for Vulnerability Analysis,* BRL Technical Report ARBRL-TR-02480, NTIS AD No. A-126-657 (March 1983).

NEWM79.

W. M. Newman and R. F. Sproul, *Principles of Interactive Computer Graphics, 2nd ed.,* McGraw-Hill, New York (1979).

OKIN73a.

N. Okino, Y. Kakazu, and H. Kubo, "TIPS-1: Technical Information Processing System for Computer-Aided Design, Drawing, and Manufacturing," *Computer Languages for Numerical Control,* pp. 141-150, North-Holland Publishing Co, Amsterdam (1973).

OKIN78.

N. Okino and et al., *TIPS-1, '77 Version,* Institute of Precision Engineering, Hokkaido University, Sapporo Japan (March 1978).

OKIN73b.

N. H. Okino, H. Kubo, and Y. Kakazu, "TIPS-1: practical approaches for an integrated CAD/CAM System," *Proceedings SME Conference on CAD/CAM,* Detroit MI (1973).

OKIN74.

N. H. Okino and et al., "Graphics processor for computer-aided design (TIPS)," *Proceedings of International Conference on Precision Engineering,* pp. 366-371 (1974).

OKIN76.

N. H. Okino, H. Kubo, and Y. Kakazu, "An integrated CAD/CAM system: TIPS-2," *Proceedings Prolomat Conference,* UK (1976).

OZOL82.

A. Ozolins and D. A. Ringers, "ISLAVE: Interactive Simple Lethality and Vulnerability Estimator," *Proceedings of the Second Joint Technical Coordinating Group for Aircraft Survivability Workshop on Survivability and Computer-Aided Design,* 1, pp. 91-99, USAF Museum, Wright-Patterson AFB, OH (18-20 May 1982).

RAPP76.

J. R. Rapp, *A Computer Model for Predicting Infrared Emission Signatures of An M60A1 Tank,* BRL Report No. 1916, NTIS AD No. B013411L (August 1976).

RAPP83.

J. R. Rapp, *A Computer Model for Estimating Infrared Sensor Response to Target and Background Thermal Emission Signatures,* BRL Memorandum Report ARBRL-MR-03292 (August 1983).

REQU82.

A. A. G. Requicha and H. B. Voelcker, "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications,* 2, 2, pp. 9-24 (March 1982).

RHOD74b.

W. A. Rhodes, *Development of a Code System for Determining Radiation Protection of Armored Vehicles (the VCS Code),* Oak Ridge National Laboratory Report ORNL-TM-4664, Oak Ridge TN (October 1974).

RIES80.

R. F. Riesenfeld, E. Cohen, and T. Lyche, "Discrete B-splines and Subdivision Techniques in Computer Aided Geometric Design and Computer Graphics," *Computer Graphics and Image Processing,* 14, 2, pp. 87-111 (October 1980).

RING81.
D. A. Ringers and F. T. Brown, *SLAVE (Simple Lethality and Vulnerability Estimator) Analyst's Guide,* BRL Technical Report ARBRL-TR-02333, NTIS AD No. B059679 (June 1981).

RITC78b.
D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "The C Programming Language," *Bell System Technical Journal,* 57, 6, pp. 1991-2019 (1978).

RITC78a.
D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Bell System Technical Journal,* 57, 6, pp. 1905-1929 (1978).

ROGE90.
D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics, 2nd ed.,* McGraw-Hill, New York (1990).

ROGE85.
David F. Rogers, *Procedural Elements for Computer Graphics,* McGraw-Hill, New York (1985).

ROTH80.
S. D. Roth, *Ray Casting as a Method for Solid Modeling,* Research Publication GMR-3466, General Motors Research Laboratories, Warren MI (October 1980).

SPUR75.
G. Spur and J. Gausemeier, "Processing of Workpiece Information for Producing Engineering Drawings (COMPAC)," *Proceedings 16th International Machine Tool Design and Research Conference,* pp. 17-21, Manchester UK (1975).

SPUR76a.
G. Spur, J. Gausemeier, and G. Muller, "COMPAC -- the use of computer internal workpiece models for design and manufacturing," *Rep. Technische Universitat Berlin,* Lehrstuhl und Institut for Werkzeugmaschien und Fertigungstechnik (1976).

SPUR76b.
G. Spur and F. L. Krause, "Stages of integration for computer supported design and manufacturing process planning (COMPAC)," *IFTOMM,* Italy (October 1976).

SPUR78.
G. Spur, "Status and Further Development of the Geometric Modelling System COMPAC," *Proceedings Geometric Modelling Project Meeting,* pp. 1-35, P-78-GM-01, CAM-I, Inc., St. Louis MO (March 14-16 1978).

STAY84.
P. R. Stay, *Rounded Edge Primitives and their Use in Computer Aided Geometric Design,* MS dissertation, University of Utah (August 1984).

SWEE86.
M. A. J. Sweeny and R. H. Bartels, "Ray Tracing Free-Form B-spline Surfaces," *IEEE Computer Graphics* (February 1986).

THOM84.
S. W. Thomas, *Modelling Volumes Bounded by B-spline Surfaces,* PhD dissertation, University of Utah (June 1984).

TILO80.
R. B. Tilove and A. A. G. Requicha, "Closure of Boolean Operations of Geometric Entities," *Computer Aided Design,* 12, 5, pp. 219-220 (September 1980).

TOOM82.
J. C. Toomay, *Radar Principles for the Non-Specialist,* Lifetime Learning Publications, London (1982).

VEEN79.
P. Veenman, "ROMULUS -- The Design of a Geometric Modeller" in *Geometric Modelling Seminar,* ed. W. A. Carter, pp. 127-152, P-80-GM-01, CAM-I, Inc., Bournemouth UK (November 1979).

VOEL81.
Voelcker, H.B. and W.A. Hunt, "The role of Solid Modelling in Machining-Process Modelling and NC Verification," *Proceedings of the 1981 SAE International Congress and Exposition,* pp. 1-8, Detroit MI (February 1981).

VOEL74.
H. B. Voelcker, A. A. G. Requicha, E. E. Hartquist, W. B. Fisher, J. E. Shopiro, and N. K. Birrell, *An Introduction to PADL: Characteristics, Status, and Rationale,* Tech. Memo. No. 22, Production Automation Project, University of Rochester (December 1974).

VOEL78.
H. B. Voelcker, A. A. G. Requicha, E. E. Hartquist, W. B. Fisher, J. Metzger, R. B Tilove, N. K. Birrell, W. A. Hunt, G. T. Armstrong, T. F. Check, R. Moote, and J. McSweeney, "The PADL-1.0/2 System for Defining and Displaying Solid Objects," *Computer Graphics (Proceedings of Siggraph '78),* 12, 3, pp. 257-263 (August 1978).

WEAV80.
E. P. Weaver and M. J. Muuss, *Interactive Construction of COM-GEOM Targets,* BRL Spring Technical Conference (1980).

WEAV82.
E. P. Weaver and P. H. Deitz, "Solid Modeling in Survivability/Vulnerability," *Proceedings of the Second Joint Technical Coordinating Group for Aircraft Survivability Workshop on Survivability and Computer-Aided Design,* 1, USAF Museum, Wright-Patterson AFB, OH (18-20 May 1982).

WHIT80.
J. T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM,* 23, 6, pp. 343-349 (June 1980).

WYVI85.
G. Wyvil and T. L. Kunii, "A Functional Model for Constructive Solid Geometry," *The Visual Computer,* 1, 1, pp. 3-14 (July 1985).

Plate Captions

1.1 ComGeom Primitive Solids
1.2 Rendered Spline Surface
1.3 AuBuC Wireframe
1.4 AuBuC Rendering
1.5 A-B-C Wireframe
1.6 A-B-C Rendering
1.7 (A-B)+C Wireframe
1.8 (A-B)+C Rendering
1.9 C-A-B Wireframe
1.10 C-A-B Rendering
1.11 Early Rendering of Vehicle

2.1 MGED Faceplate with First-level Menu
2.2 Engine Connecting Rod. Unevaluated and Evaluated Wireframes
2.3 Close-up of Connecting Rod Joint
2.4 ARB8 Before Editing
2.5 ARB8 After Editing Edge 1-4

4.1 Ray-Tracing Features
4.2 Inside and Outside Views of BMP
4.3 M109 with Transparent Armor
4.4 M48 Ray-Tracing Sample, w/Materials Colored
4.5 Main Penetrator and Spall Cone
4.6 Output of Vulnerability Evaluation
4.7 Neutron Study of Testbed Vehicle
4.8 Optical Signature Modeling
4.9 Dihedral and Trihedral Plates
4.10 Ray Paths of Radar Bounces
4.11 Ultra-high Resolution M48 SAR Image

plate	slide	hi-res	& what	where
-----	-----		-------------	-----
1.1	27	N	comgeom primitives	/demo/tanks/allsolids.rle
1.2	26	Y	rendered spline	talks/86tokyo/g/a43.rle
1.3	30	Y	AuBuC	talks/86tokyo/g/all1.rle
1.4	32	Y	AuBuC	talks/86tokyo/g/all2.rle
1.5	33	Y	A-B-C	talks/86tokyo/g/a-b-c1.rle
1.6	34	Y	A-B-C	talks/86tokyo/g/a-b-c2.rle
1.7	35	Y	A-B+C	talks/86tokyo/g/a-b+c1.rle
1.8	36	Y	A-B+C	talks/86tokyo/g/a-b+c2.rle
1.9	37	Y	C-A-B	talks/86tokyo/g/c-a-b1.rle
1.10	38	Y	C-A-B	talks/86tokyo/g/c-a-b2.rle
1.11	122	N	example of early rendering	/demo/gift/york.rle
2.1	72	Y	faceplate, first menu	talks/86tokyo/g/p2.1.rle
2.2	65	Y	two crods	talks/86tokyo/g/p2.2.rle
2.3	66	Y	close-up crod	talks/86tokyo/g/p2.3.rle
2.4	73	Y	arb8 before editing	talks/86tokyo/g/p2.4.rle
2.5	74	Y	arb8 after moving edge	talks/86tokyo/g/p2.5.rle
4.1	91	Y	ray-tracing features	/demo/lgt/raytrace-features.rle
4.2	123	N	inside/outside views of BMP	/demo/tanks/bmpei.rle
4.3	124	Y	transparent armor M109	/demo/lgt/m109,trans,front.rle
4.4	134	N	m48 side slice	/demo/tanks/m48linesZ.rle

4.5	125	Y	spall cone	/demo/lgt/m1a1_spall.rle
4.6	137	N	vulnerability evaluation (pkh)	/demo/tanks/icav.rle
4.7	138	N	testbed, neutron study	/demo/tanks/testbed.rle
4.8	142	N	optical signature modeling	/demo/lgt/osm.rle
4.9	143		dihedral and trihedral plates	˜moss/plot/bmp_flat.plot
4.10	144	N	radar bounces	/demo/sar_images/m48corners.rle
4.11	145	N	m48 sar hi-res	/demo/sar_images/m48sarHR.rle