Ballistic Research Laboratory CAD Package

Release 1.12

(1-Jan-1987)

A Solid Modeling System and Ray-Tracing Benchmark Distribution Package

SECAD/VLD Computing Consortium

The US Army Ballistic Research Laboratory

Aberdeen Proving Ground, Maryland 21005

Principle Developers

Michael Muuss
Phillip Dykstra
Keith Applin
Gary Moss
Edwin Davisson
Paul Stay
Charles Kennedy

Additional Contributions by

Douglas Gwyn
Robert Reschly, Jr.
Earl Weaver
Douglas Kingston
Bob Suckling
Jeff Hanes
Donald Merritt
Terry Slattery (USNA)
Steve Satterfield (USNA)
William Lindemann (Sun)
Steve Willson (NRTC)

Table of Contents

1. README
   - information about the BRL CAD distribution.

2. MAN PAGES

ap-pix(1) – convert Applicon files to pix files
brlcad(1) – BRL CAD programs for solid modeling, raytracing, graphics, image processing
bw-fb(1) – display a black and white bw image on a framebuffer
bw-impress(1) – convert a black and white bw image file to imPRESS bitmap
bw3-pix(1) – merge black and white bw images into a color pix file
bwcrop(1) – crop black and white bw files
bwdiff(1) – compare two black and white files
bwfilter(1) – apply a 3x3 filter to a black and white bw file
bwhist(1) – display a histogram of a black and white bw file on a framebuffer
bwhisteq(1) – equalize the histogram of a black and white bw file
bwmod(1) – arithmetically modify pixel values in black and white bw file
bwrect(1) – extract a rectangle from a black and white bw file
bwrot(1) – rotate, invert, or reverse a black and white bw file
bwscale(1) – change the size of a black and white bw file
bwstat(1) – list statistics related to a black and white bw image.
conv-vg2g(1) – upgrade old to new format binary MGED database
dunncolor(1) – set exposure values for a Dunn Model 631 camera
dunnsnap(1) – expose film in a Dunn Model 631 camera
fb-bw(1) – save a framebuffer image in a black and white bw file
fb-pix(1) – save a framebuffer image in a color pix file
fb-rle(1) – save a frame buffer image in run-length encoded form
fbanim(1) – animate small images on framebuffer using hardware pan and zoom
fbbackgnd(1) – display a shaded background on a framebuffer
fbclear(1) – clear a frame buffer image
fbcmap(1) – change the color map of a frame buffer
fbcmrot(1) – rotate the color map
fbcolor(1) – interactively pick colors
fbed(1) – framebuffer editor
fbframe(1) – place a colored frame around a framebuffer
fbgrid(1) – draw a grid on a frame buffer
fbpoint(1) – point at and identify screen coordinates
fbscanplot(1) – plot the intensities of a framebuffer scanline
fbzoom(1) – enlarge a frame buffer image
g2asc(1) – convert between binary and ASCII MGED databases.
gencolor(1) – output a pattern of byte values
loop(1) – generate integer indices for shell loops, etc
mged(1) – multi-display interactive combinatorial solid geometry editor
pix-bw(1) – combine pix color planes into a bw black and white image
pix-bw3(1) – split up the color planes of a color pix file
pix-fb(1) – display a pix format RGB image file on a framebuffer
pix-rle(1) – convert a pix format RGB file into run-length encoded form
pix2asc(1) – pix2asc, asc2pix - encode and decode color pixel files
pixbustup(1) – break up concatenated pix images into seperate files
pixdiff(1) – highlight pixel colors which differ between two files

pixfilter(1)	– apply a 3x3 filter to a color pix file
pixhist(1)	– display a histogram of a color pix file on a framebuffer
pixhist3d-pl(1)	– output RGB color space utilization in 3D unix-plot
pixhist3d(1)	– display RGB color space utilization
pixinterp2x(1)	– interpolate a color pix file twice as large as the original
pixrect(1)	– extract a rectangle from a color pix file
pixrot(1)	– rotate, invert, or reverse a color pix file
pixscale(1)	– change the size of a color pix file
pixstat(1)	– list statistics related to a color pix image
pixtile(1)	– paste small images together into one large image
plot-fb(1)	– display 3-D UNIX plot on a framebuffer
pp-fb(1)	– display a GIFT pretty-picture file on a frame buffer
rle-fb(1)	– decode run-length encoded image onto a frame buffer
rle-pix(1)	– restore run-length encoded image and convert to RGB pixel format
rt(1)	– raytrace an mged model, giving a shaded picture
rtpp(1)	– raytrace an mged model, writing a GIFT-style Pretty Picture file
rtray(1)	– raytrace an mged model, writing a VLD rayfile
libfb(3)	– multiple device, generic frame buffer library
libplot3(3)	– graphics interface subroutines
librle(3)	– run-length encoded image library
librt(3b)	– library for raytracing an MGED database
bw(5)	– Black and White image file format and utilities
pix(5)	– Color RGB image file format and utilities
plot3(5)	– 3-D and color graphic plotting interface

3. Understanding the Preparation and Analysis of solid Models

4. GED: An Interactive Solid Modeling System for Vulnerability Assessments

5. MGED: Multi-device Graphics Editor

1. README

BALLISTIC RESEARCH LABORATORY

SOLID MODELING SYSTEM
and
RAY-TRACING BENCHMARK

DISTRIBUTION PACKAGE
Release 1.12
(1-Jan-87)

DISTRIBUTION DETAILS

To obtain a copy of the distribution, you should send the authors:

1) Enough magnetic tape for 12 Mbytes of data. For 9-track 1/2 inch magtape, specify density (800, 1600, or 6250). Silicon Graphics and SUN tape cartridges can also be accommodated. Specify TAR or CPIO, and blocksize.

2) A letter indicating:

   -- Who you are
   -- What the BRL CAD package is to be used for
   -- Equipment and operating system(s) you plan on using
   -- That you agree to the conditions listed below.

The software is distributed free of charge with the following conditions:

1. The BRL CAD package source files won't be passed on to third parties. If someone wants them, have them contact BRL. We need to know who has what, and what it is being used for.

2. BRL will be credited should the software be used in a product or written about in any publication. BRL will be referenced as the original source in any advertisements.

3. BRL assumes no legal responsibility for source code and its subsequent use. No warranty is expressed or implied.

4. If any bugs or problems are found they will be reported back to BRL.

You will own full rights to any databases or images you create with this package.

To obtain authorization for use at additional sites, or for permission to use this work as part of a commercial package, please contact the authors at the address below.

CONTENTS

What you have here is a collection of software which includes:

libsysv	Some System-V compatability routines
mged	A solid-model editor
librt	A solid-model ray-tracing library
rt	A ray-tracing lighting model, for rendering
db	Several solid-model databases, in ASCII form

conv	ASCII/binary database converters
bench	Scripts to drive the RT benchmark
pix	Reference images for the RT benchmark, in ASCII form
libpkg	A "message-passing" interface to TCP network links
libfb	A generic frame-buffer library
rfbd	TCP server for remote frame-buffer access
libtermio	A library to handle terminal mode setting
libplot3	A public-domain 2-D and 3-D UNIX-Plot library
librle	A Run-Length-Encoding library (originally from UofUtah)
util	Zillions of image-handling utilities, as tools
fbed	Frame-buffer image editor
vdeck	Convert mged models to GIFT-format card decks.
dmdfb	libfb support for layers in Teletype 5620 DMD terminal

This distribution does NOT include the various BRL-specific model analysis tools such as GIFT, SAR, SLAVE, VAST, etc.

For overview information on the whole package, consult the brlcad(1) manual page, in the util/ directory. For a discussion of the significance of this software, read the paper "Understanding the Preparation and Analysis of Solid Models", included in the paper/ directory.

The first sections of this file discuss how to configure and compile just enough software to run the benchmark. After the benchmark results are presented, information about configuring and installing the full software package is presented. If you encounter difficulty with running the benchmark, you might wish to build the full package, to take advantage of the image tools in tracking down your trouble.

BENCHMARK BACKGROUND

This file describes how to set up enough of the CAD package to run a benchmark test of the RT ray-tracing program on the sample databases provided. Each run will produce a 512x512x24 bit color shaded image in a "bench/xxx.pix" file, and a run log in "bench/xxx.log". Reference copies of are provided as "pix/xxx.pix" and "pix/xxx.log". The sample databases provided are processed in order of increasing difficulty, and each produces a colorful image. In each picture there are two light sources; the primary source is located near the center of the model (the white ball), and the secondary light source is located at the "eye" position.

moss.g -- This is a simple model, containing a yellow torus, a green ellipsoid, a bluish cube, a pink truncated (distorted) cone, all sitting on a blue plate. Note the shadowing, and "specular splash".

world.g -- The same model as moss.g, but with various rendering features enabled. The model is enclosed in a hollow cloud sphere. The plate is a mirror, the egg is crystal, the box has a debugging texture map applied.

star.g -- A familiar sight to Star Trek fans; a low-detail exterior of the ship "Enterprise". The hull is white, with some other structures in other colors. The small red object in the foreground is the shuttlecraft Galileo, to scale.

bldg391.g -- An exploded view of a two-story imaginary computer site, including various walls, red stairways, green disk drives, etc. The lower floor is a mirror, the upper floor is glass.

The RT benchmark requires a binary database for input ("pix/xxx.g"), and produces a binary image as output ("bench/xxx.pix"), along with a log file ("bench/xxx.log"). To make this distribution machine independent, both the databases and the reference copies of the images are provided in an ASCII format in the "db" and "pix" directories, and ascii-to-binary converters are provided. If you are running this benchmark on a non-UNIX machine that has difficulty writing binary streams, change the "-o file.pix" flag to "-O file.a_pix" and RT will write the ASCII form directly. For more discussion, consult the doc/pix.5 and conv/asc2pix.1 manual pages.

BENCHMARK CONFIGURATION

1) The first configuration step is to edit the "Makefile.defs" file in the top directory of the distribution. In a future version of the distribution, the main alternatives for this file will all be provided; right now, a variety of things need to be tailored. Everything is clearly marked; #SYSV# appears near symbols that need to be changed to run on a System V UNIX. Certain vendor-specific settings are similarly marked, eg, #SGI#, #GOULD#, #ALLIANT#, #CRAY#, etc. As distributed, this file is configured for a 4.n BSD Berkeley UNIX system on a VAX.

Select the correct value for SYSTEM.

At present, SYSVERS should be defined only on Silicon Graphics (SGI) systems. Set SYSVERS to either SGI34 for Release 3.4 and below, or SGI35 for Release 3.5 and above. If you do not have the TCP option on your SGI, don't define either.

Configuration of libfb. Full details on configuring the framebuffer library are provided in the "Full Configuration" section, below. If you just desire to run the benchmark but not display images, this library is still necessary, as the debug and disk-file interfaces are always provided. In this case, the LIBFB_CFILES, LIBFB_OBJS, and LIBFB_CONFIG lines should be null lists, ie:

```
LIBFB_CFILES=
LIBFB_OBJS=
LIBFB_CONFIG=
```

In the case of extreme difficulty, a noop-stub version of libfb is provided in rt/libfb-dummy.c.

The timer configuration LIBRT_TIMER must be specified, based on the version of UNIX you are using. This timer module is for providing CPU and elapsed time indications. If your system has a vendor-specific clock with more resolution than the normal UNIX library provides, you should create your own system-specific timerXXX.c module. The line is distributed as:

```
LIBRT_TIMER = timer42
```

For System V, select instead:

```
LIBRT_TIMER = timerunix
```

The full range of choices are:

timer42	4.2 BSD, 4.3 BSD (Berkeley UNIX)
timerunix	Generic UNIX, System III, V, etc.
timer52brl	For BRL's System 5-under-4.n BSD
timerhep	Denelcor HEP
timercray	Cray X-MP running UNIX, and Cray-2
timercos	Cray X-MP running COS

2) The file "h/machine.h" has a number of configuration options, including the setting of:

fastf_t	fastest "C" floating point type (**float** or **double**)
LOCAL	**static** or **auto**, for fastest local data type.
FAST	LOCAL or register, for FAST fastf_t variables.
PARALLEL	Defined for parallel processors.

VAX, GOULD, Cray, Alliant and HEP examples are provided. The defaults should be correct for reasonable 32-bit serial computers. Note that K&R "C" specifies that all floating point calculations must be done as **double**, so it is usually fastest to set fastf_t to **double**. Also note that when **float** is 32-bit IEEE or IBM style floating point, not enough accuracy is carried to correctly process large models. See the "Silicon Graphics Warnings" section for more information.

3) The file "rt/Makefile.loc" MUST be edited for the benchmark run, to turn off various random effects, which if left enabled would make comparison to reference files impossible. Be certain to uncomment the line which reads:

```
CONFIGDEFS = -DBENCHMARK
```

4) It is important to run "make depend" in the top directory of the distribution each time these Makefiles are changed. For libfb, mged, and librt, it is also necessary to run "make clean" in those directories after making related changes to the Makefiles, because the binaries of some modules depend on configuration options selected in the Makefiles.

THE BENCHMARK.

These steps are involved in loading the software and running the benchmark on a UNIX system:

1) Load the distribution.

```
mkdir cad
cd cad
tar x
```

(On System V, use "tar xo" instead, to forgo chown-ing away the files).

2) Edit the various header files and Makefiles.

```
ed Makefile.defs
ed h/machine.h              # defaults OK for 32-bit systems
```

3) Compile everything, and convert the databases and reference images.

```
make benchmark
```

4) Run the benchmark. Make sure that the answers match the reference files to within plus-or-minus one in each color (see pixdiff(1)).

```
cd bench
sh run          # takes several VAX hours.
cd ..
```

5) Compare the execution times in the "bench/xxx.log" files with the VAX-11/780 (with hardware FPA) times which are given in the "pix/xxx.log" files. The significant figure is the "rays/sec" figure (the "RT Figure of Merit", RTFM) and should only be compared between different runs on the same model.

BENCHMARK RESULTS

In the tradition of Dongarra, the Rays/sec figure is considered the "RT Figure of Merit" (RTFM). Note that the RTFM can only be compared between different runs on the *same* database; it is not meaningful to compare RTFM numbers between different databases. The Rays/sec number for a multi-processor machine is for "aggregate CPU cluster seconds", rather than rays/total CPU seconds (which remains fairly constant on good parallel machines). Worthy finalists to date are listed below. Entries are omitted for tests in which correct results were not obtained.

System Name	Configuration
nrtc-gremlin	ISI 68020, no FP hardware, 4.2 BSD
brl-virus	Sun-3/50, w/68881, Sun UNIX 3.2
vortac	Sun-3/160, 16.67 Mhz clock, 12 Mhz 68881, Sun 3.2
brl-vista	SGI 3030, w/FP chip, UNIX release 3.5
brl-vat	VAX 750, FPA, 4.2 BSD
brl-vgr	VAX 780, FPA, 4.3 BSD
nrtc-785	VAX 785, FPA, VMS+Eunice_4.1
nrtc-dryer	Ridge 330, ROS
ardec-3	Pyramid 90Mx, Dual-CPU, OSx 2.5, 16 Mbytes
brl-spark	Gould PN 9050, no MACC, UTX 1.2, 2x4Mb mem boards
brl-vmb	Gould PN 9080, no MACCs, UTX 2.0, 16x1Mb mem boards
amsaa-seer	Gould PN 9080, w/MACCs, UTX 2.0, 4x4Mb mem boards
brl-vector	Alliant FX/8 (4 CEs, 6 IPs, 16 Mbytes) Concentrix 2.0
nrl-xmp	Cray X-MP/12, COS V115, 9.5ns clock
snm2	Cray 1-M, SN2, UNICOS 2.0
brl-patton	Cray X-MP/48, SN213, COS V115BF2, 8.5ns clock

FILE: "bench/moss.log"			
Pix/sec	Ray/sec	*780	Machine & OS
12.3			nrtc-gremlin
57.2	81.6	0.73	brl-vat
78.5	112.1	1.00	brl-vgr
81.5	116.4	1.04	brl-vista
86.7			nrtc-785
89.2	127.3	1.14	brl-virus
103.8	148.2	1.32	vortac
158.6	226.5	2.02	nrtc-dryer
269.4	384.7	3.43	brl-spark, UTX 1.2
289.5	413.4	3.69	brl-spark, UTX 2.0
289.9	413.9	3.69	brl-vmb, UTX 2.0
332.6	460.7	4.11	amsaa-seer, UTX 2.0
345.2	492.9	4.40	brl-vector, 1 CE, no vectors (1.00X)
679.8	904.9	8.07	brl-vector, 2 CEs, no vectors (1.84X)
1018.2	1375.5	12.27	brl-vector, 3 CEs, no vectors (2.79X)
1309.5	1813.5	16.17	brl-vector, 4 CEs, no vectors (3.68X)
1396.6			nrl-xmp, no vectors, no optim
1745.0	2492.9	22.24	snm2, no vectors, no optim
2418.3	3453.3	30.81	brl-patton, 1 CPU, no vectors, no optim (1.00X)
4802.1	6856.5	61.16	brl-patton, 2 CPUs, no vectors, no optim (1.99X)
7148.1	10205.1	91.04	brl-patton, 3 CPUs, no vectors, no optim (2.96X)
9331.8	13320.2	118.82	brl-patton, 4 CPUs, no vectors, no optim (3.86X)

FILE: "bench/world.log"			
Pix/sec	Ray/sec	*780	Machine & OS
35.9	63.6	1.00	brl-vgr
38.5	68.4		ardec-3
39.4	69.9		brl-virus
45.6	81.0		vortac
48.4	86.0		brl-vista
128.5	228.0		brl-vector, 1 CE, no vectors
130.7	232.0		brl-spark, UTX 2.0
131.5	233.4		brl-vmb, UTX 2.0
148.3	263.2		amsaa-seer, UTX 2.0
257.3	424.6		brl-vector, 2 CEs, no vectors
379.1	650.0		brl-vector, 3 CEs, no vectors
491.3	845.5		brl-vector, 4 CEs, no vectors
853.3	1514.3		brl-patton, 1 CPU, no vectors

FILE: "bench/star.log"			
Pix/sec	Ray/sec	*780	Machine & OS
46.0	55.0	1.00	brl-vgr
47.7	57.0		brl-virus
47.8	57.2		ardec-3
55.1	65.8		vortac
150.6	180.0		brl-vector, 1 CE, no vectors (1.00X)
176.2	210.6		brl-vmb, UTX 2.0
178.8	213.8		brl-spark, UTX 2.0
206.5	246.8		amsaa-seer, UTX 2.0
294.1	349.3		brl-vector, 2 CE, no vectors (1.94X)
443.4	523.4		brl-vector, 3 CE, no vectors (2.91X)
577.9	686.0		brl-vector, 4 CEs, no vectors (3.81X)
1063.7	1271.2		brl-patton, 1 CPU, no vectors (1.00X)
2110.6	2522.2		brl-patton, 2 CPUs, no vectors (1.98X)
3138.2	3749.9		brl-patton, 3 CPUs, no vectors (2.95X)
4147.1	4955.6		brl-patton, 4 CPUs, no vectors (3.90X)

FILE: "bench/bldg391.log"			
Pix/sec	Ray/sec	*780	Machine & OS
34.1	50.7	1.00	brl-vgr
35.2	52.4	1.03	brl-virus
40.9	60.7	1.20	vortac
68.8	102.3	2.02	brl-vista
105.7	157.1	3.10	brl-vector, 1 CE, no vectors (1.00X)
142.6	211.9	4.18	brl-spark, UTX 2.0
143.0	212.6	4.19	brl-vmb, UTX 2.0
162.2	241.0	4.75	amsaa-seer, UTX 2.0
210.6	312.6	6.17	brl-vector, 2 CEs, no vectors (1.98X)
310.9	459.9	9.07	brl-vector, 3 CEs, no vectors (2.93X)
405.2	600.7	11.85	brl-vector, 4 CEs, no vectors (3.82X)
691.2	1027.3	20.26	brl-patton, 1 CPU, no vectors

It would be greatly appreciated if you would return to BRL any statistics that you gather, including the manufacturer of the machine, hardware and software version numbers, presence or absence of floating point hardware, and the name (preferably InterNet or UUCP) of the specific machine used.

FURTHER OPTIMIZATIONS

After initial testing, you may wish to go back and enable the special assembler version of sqrt(), by uncommenting and editing the FAST_SQRT line in "librt/Makefile.loc". 4.2BSD VAX and Gould UTX 1.2 versions are provided.

The file "h/vmath.h" has macros for numerous common vector operations; if your system has hardware support for vector operations, after getting the program working you would profit greatly from changing these macros to machine-specific code. The most important macros are MAT4X3VEC and MAT4X3PNT. If you do this, please submit the "rays/sec" figures for the original as well as the optimized runs.

BRL is currently attempting explicit vector extensions to the C language; it is anticipated that this work will be included in the next release.

SILICON GRAPHICS WARNINGS

If you are attempting to run the benchmark on an SGI IRIS, beware of the **long float** -vs- **double** controversy. The SGI C compiler converts both **float** and **double** to 32-bit single-precision numbers. While the current release of librt is believed to generally operate correctly in spite of this, beware of the RT program failing to find roots, or being unable to perform a "box push". Unfortunately, the "star.pix" RT benchmark will will fail due to lack of floating point precision, and generate lots of diagnostics, filling your disk.

SGI software release version 3.3.1, 3.4, and 3.5 have a defective library version of hypot(); be aware of the (slower) substitute in rt.c

In SGI release 3.5, printf() is unable to print the numbers 500.0, 1500.0, 2500.0, etc., when using the hardware floating-point chip (eg, -Zf compiler flag) due to a bug in the hardware divide causing troubles in modf(). This will cause 500.0 to print as "4:0.0". This problem goes away if you remove the -Zf flag when compiling, but then your speed vanishes as well. A special hack exists in conv/Makefile.loc if you have problems converting databases, but the trouble runs deeper than that.

Operations of the form "*fp += incr;" are known to compile incorrectly on release 3.5 when "fp" is declared as "**register double** *fp" (or **register float**). The one known instance of this in util/bwmod.c has been "fixed" with a **#ifdef** sgi.

The MGED big-E command (display evaluated boundaries) is known to dump core on the SGI on many databases. This has not been isolated yet.

Currently there are no good solutions to these problems. Voice your opinions to the vendor.

FULL CONFIGURATION

The following assumes that you have already loaded the distribution per step 1 of "The Benchmark" section, and performed the configuration outlined in the "Benchmark Configuration" section, above.

You may wish to install the package in places other than BRL does. In general, all CAD executable programs are installed in BINDIR (/usr/brl/bin), with libraries going in LIBDIR (/usr/brl/lib). Users are told to find the CAD-related include files in INCLUDE_DIR (/usr/include/brlcad). Be certain to examine these defines in Makefile.defs. The remote-frame-buffer daemon is installed in /usr/brl/etc, see rfbd/Makefile.loc.

The configuration for libfb must be edited to reflect the display hardware that you have. Even if your system has no display hardware, this library is necessary, as the debug and disk-file interfaces are always useful. (Lacking any displays at all, you may be able to utilize the bw-impress(1) program to produce pictures on an Imagen laser-printer). In the case of no display hardware at all, the LIBFB_CFILES, LIBFB_OBJS, and LIBFB_CONFIG lines should be null lists, ie:

```
LIBFB_CFILES =
LIBFB_OBJS   =
LIBFB_CONFIG       =
```

If this is an SGI system of release 3.5 with TCP, then certain kernel limits need to be adjusted for libfb/if_sgi.c to work well. See the detailed comment on the front of that module. The display is configured by:

```
LIBFB_CFILES = if_sgi.c
LIBFB_OBJS   = if_sgi.o
LIBFB_CONFIG       = -DIF_SGI
```

If this is an SGI system of release 3.4 or earlier with Excelan TCP support, the display can be supported by:

```
LIBFB_CFILES = if_sgi.c
LIBFB_OBJS   = if_sgi.o sgiselect.o
LIBFB_CONFIG       = -DIF_SGI
```

If this is a TCP-capable system, you will want to include the remote frame-buffer capability; the defaults already provide for this. If you are using a system which does not have TCP capability, you must edit the top-level Makefile and remove the "needed for IF_REMOTE" line in the "benchmark" rule, to prevent compiling "libpkg", in addition to configuring in Makefile.defs. For no remote capability, be certain that IF_REMOTE *et. al.* are not defined in the LIBFB_CONFIG lines. Build a remote-framebuffer-only library with:

```
LIBFB_CFILES = if_remote.c
LIBFB_OBJS   = if_remote.o ../libpkg/pkg.o
LIBFB_CONFIG       = -DIF_REMOTE
```

To offer in-bound remote framebuffer service, you may wish to compile "rfbd", the remote-framebuffer daemon. It can be started either from /etc/rc.local, or via inetd(8) with this /etc/inetd.conf entry:

```
remotefb stream tcp    nowait  nobody  /usr/brl/etc/rfbd      rfbd
```

Also, be certain to add this line to /etc/services:

```
remotefb       5558/tcp                    # Remote Frame Buffer
```

As distributed, the remote, Adage, and Pseudo-tty (Teletype DMD 5620) interfaces are all enabled. You will almost certainly wish to change this; failure to do so is the most common source of difficulties in installing this distribution. If you change the configuration of libfb partway through building the system, be certain to run "cd libfb; make clean; make depend" before recompiling, because the Makefiles and binaries change based on the configuration options in Makefile.defs.

The MGED (multi-device model editor) needs to be configured. At present, all versions of MGED have support for these types of display devices by default:

plot	any UNIX-plot filter
vg	Vector General 3300
mg	Megatek 7250
tek	Tektronix 4014 and family
tek4109	Tektronix 4109

These optional display devices are also supported, when specifically configured:

ir	SGI IRIS 2400, 3030
sunpw	SUN 3.0 Pixwin
rat	Raster Tech One/80
ps	E&S Picture System 300
mer	Megatek Merlin 9200 [Marginal]
sab	Saber w/GOPS [Marginal]

Necessary vendor-supplied libraries must be listed on the MGED_LIBES line.

To configure just the default set of displays, specify:

```
MGED_CFILES=
MGED_OBJS  =
MGED_CONFIG        =
MGED_LIBES  =
```

To configure an SGI Iris ("ir") running 3.5, specify:

```
MGED_CFILES= dm-ir.c
MGED_OBJS  = dm-ir.o
MGED_CONFIG        = -DDM_IR
MGED_LIBES  = -lbsd -lgl2
```

To configure an SGI Iris ("ir") running 3.4, specify:

```
MGED_CFILES= dm-ir.c
MGED_OBJS  = dm-ir.o
MGED_CONFIG        = -DDM_IR
MGED_LIBES  = ../libfb/sgiselect.o -lgl2
```

To configure a SUN, specify:

```
MGED_CFILES= dm-sunpw.c
MGED_OBJS  = dm-sunpw.o
MGED_CONFIG        = -DDM_SUNPW
MGED_LIBES  = -lsuntool -lsunwindow -lpixrect
```

Also be certain to properly define the MGED_VERS variable.

If this is a System V machine, you will also have to edit the top level Makefile to remove rfbd, libpkg, and fbed from the PRODUCTS list. Also, you will have to edit util/Makefile.loc and remove dunncolor and dunnsnap from the PRODUCTS list.

FULL INSTALLATION

The steps involved in installing the full BRL CAD package on a UNIX system are these:

1) Load the distribution.

```
mkdir cad
cd cad
tar x
```

(On System V, use "tar xo" instead, to forgo chown-ing away the files).

2) Edit the various header files and Makefiles.

```
ed Makefile.defs
ed h/machine.h          # defaults OK for 32-bit systems
ed Makefile                 # to remove IF_REMOTE
ed rt/Makefile.loc          # remove BENCHMARK flag
ed rfbd/Makefile.loc        # configure rfbd
ed util/Makefile.loc        # if SYSV
```

3) Compile everything.
   make all

4) After everything compiles, install everything.
   make install

IMAGE DISPLAY

If you have a suitable framebuffer, you will probably wish to display the images generated by the benchmark, and compare them to the reference images. A pix(5) file is a sequence of **unsigned char** triples written RGB in the range 0 to 255. The first pixel goes in the lower left corner of the screen, and subsequent pixels write to the right; subsequent scan lines are displayed above earlier ones. (First quadrant display). See doc/pix.5 for more details.

Programs for dealing with images are in the "util" directory. To display a pix(5) file on a framebuffer, set environment variable FB_FILE (see brlcad(1) for details), and run:

```
cd bench
pix-fb moss.pix
```

If the images computed on your machine do not match the reference images, the program "util/pixdiff" will compute a pix file that will highlight the differences for you, and report a summary of bytes equal, off-by-1, and off-by-many, eg,

```
pixdiff moss.pix ../pix/moss.pix  pix-fb
```

For a display of the relative magnitude of the differences at each pixel, use "util/bwdiff" instead, eg,

```
bwdiff moss.pix ../pix/moss.pix  pix-fb
```

or

```
bwdiff moss.pix ../pix/moss.pix  \
bwmod -s 128 -m 4 -a 128  pix-fb
```

and for statistics on the differences, use

```
bwdiff moss.pix ../pix/moss.pix  pixstat
```

ENHANCEMENTS, COMMUNICATION

It is expected that new releases of this software will be issued roughly semi-annually. Information about new releases will be routinely provided to recipients of this software. If your address changes, please let us know, so we can update our records.

You are invited to participate in the BRL CAD software mailinglist, called <CAD @ BRL.ARPA>. Bug reports and discussions of new features are the main topics; volume of messages has been light (so far). Direct your bug reports to this address. Request a

subscription by sending to <CAD-REQUEST @ BRL.ARPA>.

If you find bugs, please report your experiences. While BRL makes no offer of support, we are most interested in hearing about your experiences. Electronic mail is vastly preferred over telephone contact. In dire circumstances, you may phone; 1500-1800 Eastern time is best.

If you develop additional software for the BRL CAD environment that you would be willing to share, please send it to us for inclusion in the next release.

Best Wishes,
-Mike Muuss

Leader, Advanced Computer Systems Team
U.S. Army Ballistic Research Laboratory
APG, MD. 21005-5066
USA

ArpaNet: <Mike @ BRL.ARPA>

Commercial (301)-278-6678
AUTOVON 298-6678
FTS 939-6678

2. MAN PAGES

NAME

ap-pix – convert Applicon files to pix files

SYNOPSIS

ap-pix [–**v**] file.ap > file.pix

DESCRIPTION

Ap-pix converts an *Applicon* drum plotter image file into a *pix*(5) file.

The *Applicon* file is assumed to consist of a number lines each 3456 bits long, the resulting *pix* file thus has scanlines of 3456 pixels.

SEE ALSO

brlcad(1), pix(5).

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

brlcad – BRL CAD programs for solid modeling, raytracing, graphics, and image processing

DESCRIPTION

brlcad is good. *brlcad* is free!

The BRL CAD Distribution contains many directories of materials. The major categories of programs are:

libsysv
Some System-V compatability routines

mged A multi-device interactive editor for constructing and updating combinatorial solid geometry (CSG) models. See *mged*(1), and *Understanding the Preparation and Analysis of Solid Models* by Michael John Muuss (enclosed, in the "paper") directory.

librt A library of functions suitable for interogation of a CSG solid model, utilizing ray-tracing techniques. See *librt*(3).

rt A ray-tracing lighting model, for rendering pictures of *mged*(1) CSG models. See *rt*(1), *rtray*(1), *rtpp*(1).

conv Programs to convert *mged*(1) and *pix*(3) format binary files to a portable ASCII form, and back again. See *asc2g*(1), *asc2pix*(1), *conv-vg2g*(1), *g2asc*(1), *pix2asc*(1).

libpkg A "message-passing" interface layered on top of TCP network links, to ease construction of network-distributed applications.

libfb A generic frame-buffer library which includes support for a number of display devices, as well as file, network, and debugging interfaces. Most programs which use this library have the letters "fb" in their names. To override your system default frame-buffer, the environment variable **FB_FILE** can be set. In addition, some commands also support a **–F** *framebuffer* option. Note that the disk file format of *libfb* is that of *pix*(5), allowing "framebuffer" files to be later processed by any of the "pix" family of programs. See *libfb*(3), *pix-fb*(3), *pix*(5), etc.

rfbd TCP-based network server which implements remote frame-buffer services.

libtermio
A library to handle terminal mode setting on both Berkeley or SystemV machines.

libplot3
A public-domain version of the UNIX-Plot library which differs from that of the standard *libplot*(3), only by the addition of 3-D primatives, color, and the use of a file pointer parameter. See *libplot3*(3), *plot3*(5).

librle A Run-Length-Encoding (RLE) library, providing functions originally from the University of Utah in a library package. Note that the current version of this library reads Edition-1 and Edition-2 RLE files. The University of Utah is currently preparing Edition-3 of this format. See *rle-fb*(1), *fb-rle*(1), *rle-pix*(1), *pix-rle*(1), *librle*(3).

util A collection of image-handling utilities, each constructed as individual tools intended to be used in combination. See *ap-pix*(1), *bw-fb*(1), *bw-impress*(1), *bw3-pix*(1), *bwcrop*(1), *bwdiff*(1), *bwfilter*(1), *bwhist*(1), *bwmod*(1), *bwrect*(1), *bwrot*(1), *bwscale*(1), *bwstat*(1), *dunncolor*(1), *dunnsnap*(1), *fb-bw*(1), *fb-pix*(1), *fb-rle*(1), *fbanim*(1), *fbbackgnd*(1), *fbclear*(1), *fbcmap*(1), *fbcmrot*(1), *fbcolor*(1), *fbframe*(1), *fbgrid*(1), *fbpoint*(1), *fbzoom*(1), *gencolor*(1), *loop*(1), *pix-bw*(1), *pix-bw3*(1), *pix-fb*(1), *pixbustup*(1), *pixdiff*(1), *pixfilter*(1), *pixhist*(1), *pixhist3d-pl*(1), *pixhist3d*(1), *pixinterp2x*(1), *pixrect*(1), *pixrot*(1), *pixscale*(1), *pixstat*(1), *pixtile*(1), *plot-fb*(1), *pp-fb*(1), *rle-fb*(1), *rle-pix*(1), *bw*(5), *pix*(5), *plot3*(5).

fbed An interactive, termcap-based frame-buffer image editor. See *fbed*(1).

vdeck A program to convert *mged(1)* models to *gift(1)* format card deck files. See *vdeck*(1).

FB_FILE

Whenever a framebuffer is used, the environment variable **FB_FILE** is checked for the device to use. The format of this variable is either *[hostname:]*/**dev**/*device_name[num]* or is the pathname of a disk file to be used as a "virtual framebuffer." *Hostname* is the name of a remote machine if the remote framebuffer interface is being used. This prefix is not used for local display devices. *Device_name* is a special string which is used to select a particular type of framebuffer. *Num* is type dependent and can either mean a display number or select some options for that type. One special *device_name* will always exist called **debug**, which displays the calls made to the framebuffer library on your terminal.

If **FB_FILE** is not set, the default for you system will be used.

The use of **/dev/** before the *device_name* is simply to distinguish them from filenames. See the end of the *libfb*(3) manual page for a list of the device names and the meanings of any *num* parameters they may take.

OPTIONS

A convention exists for the options used on most of the utilities provided. Random options are usually lower case. Options which could specify either a screen or file size are, by convention, lower case for file information, and upper case for screen information. Here's a list of some of the common options you may encounter:

−h The "high resolution" flag, increasing the default screen and file size of 512x512 to 1024x1024. Has same effect as **−s 1024 −S 1024** but exists as a convienient shorthand held over from the simple days when framebuffers where square, and only came in two resolutions. This historical usage unfortunately prempts this letter from use as a height specifier, forcing that function to relocate under protest to **−n**, which can be thought of as "number of scanlines", which isn't too bad a mnemonic.

−i Inverse flag. Pretend origin is in upper left corner of screen, for that good old-fashioned fourth-quadrant behavior.

−c Various. Typically means clear the screen first.

−z Various. Typically means zoom-in on current area of display.

−s *square_file_size*

WARNING: Behavior is undefined when the **−s** flag is used in conjunction with the BR −w or −n flags.

−w *file_width*

−n *file_height*

−S *square_screen_size*

WARNING: Behavior is undefined when the **−S** flag is used in conjunction with the BR −W or −N flags.

−W *screen_width*

−N *screen_height*

−x *file_x_offset*

−y *file_y_offset*

−X *screen_x_offset*

−Y *screen_y_offset*

TAPE FORMAT

When dealing with large collections of images, as might be needed for making a movie, it frequently becomes desirable to deal with magnetic tapes. Some of the early **pix**(5) tools contained built-in knowledge of the tape format. While this aberrant early design has been corrected in favor of using tape-oriented programs such as **dd**(1) in pipelines with the image tools, our "standard" image format remains.

Regardless of image resolution, all tape records are 24k bytes long.

For the various combinations of density and image resolution, a convention for the number of frames/reel exists:

Density	Resolution	Frames/reel
6250	1k	48
6250	512	192
1600	1k	12
1600	512	48

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bw-fb – display a black and white bw image on a framebuffer

SYNOPSIS

bw-fb [**–hiczRGB**] [**–F** framebuffer] [**–s** squarefilesize] [**–w** file_width] [**–n** file_height] [**–S** squarescrsize] [**–W** scr_width] [**–N** scr_height] [**–x** file_xoff] [**–y** file_yoff] [**–X** scr_xoff] [**–Y** src_yoff] < file.bw

DESCRIPTION

Bw-fb reads a *bw*(5) format file from standard input, and displays the image on the currently selected framebuffer. The environment variable *FB_FILE* specifies the current framebuffer, see *brlcad*(1). Alternatively, the framebuffer may be explicitly specified by using the **–F** flag.

By default, the *bw* file, and the requested framebuffer size, is assumed to be 512x512 pixels. Specifying the **–h** flag changes both of these sizes to 1024x1024.

If the **–i** flag is specified, the image is output from top to bottom. Customarily, *bw* files are stored bottom to top, so this inverts the image.

The **–c** flag causes the screen to be cleared before the image is displayed. The default behavior is to overwrite the current image with the new image without clearing the screen.

The **–z** flag will zoom and center on the image being displayed to make it fill the display area.

The flags **–R**, **–G**, and **–B** select individual color planes into which the image will be loaded. One or more can be given. By default all three colors are written with the *bw* pixel value.

The **–w** file_width flag specifies the width of each scanline in the input file, in pixels. **–W** scr_width does the same for the display device.

The **–n** file_height and **–N** scr_height flags specify the height in scanlines of the input file or display device respectively.

–s squarefilesize and **–S** squarescrsize set both the height and width to the size given.

–x file_xoff, **–y** file_yoff, **–X** scr_xoff, and **–Y** scr_yoff will offset into the file or onto the screen by the given amounts. The coordinate system for these offsets is first quadrant, with the origin at the lower left corner of the image.

FORMAT

A *bw*(5) file contains sequences of pixels. Each pixel is stored as one **unsigned char** The first pixel in a *bw* file is the lower left corner of the image. The pixels proceed from left-to-right across each scanline, with scanlines being written from the bottom to the top of the image.

SEE ALSO

fb-bw(1), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bw-impress – convert a black and white bw image file to imPRESS bitmap

SYNOPSIS

bw-impress [**–Dh**] [**–s** squaresize] [**–w** width] [**–n** height] [**–t** thresh] < file

DESCRIPTION

Bw-impress converts an 8-bit monochrome *bw*(5) format *file* (read from the standard input if not specified) to *Imagen imPRESS* language codes for plotting a gray-scale image on a 300dpi *Imagen* laser printer.

The gray level is plotted on the *Imagen* as a small pattern of dots, by default using "ordered dither". The **–D** option causes the gray levels to instead be plotted using halftones, as in newspaper/magazine reproductions of photographs. The **–h** option specifies a 1024x1024 image rather than the default of 512x512.

CAVEATS

Bw-impress will take several CPU-minutes (on a VAX 780) to convert a single *file*.

EXAMPLES

The following invocations are fairly typical:

pix-bw -ntsc image.pix bw-impress qpr –qi300 &

rle-pix image.rle pix-bw -ntsc bw-impress qpr –qi300 &

SEE ALSO

rle-pix(1), pix-bw(1), bw(5), pix(5)

DIAGNOSTICS

Error messages are intended to be self-explanatory.

AUTHORS

Geoffrey Cooper, Imagen Corporation
Douglas A. Gwyn, BRL
Michael John Muuss, BRL

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bw3-pix – merge black and white bw images into a color pix file

SYNOPSIS

bw3-pix redinput greeninput blueinput > file.pix

DESCRIPTION

Bw3-pix merges (combines the colors of) *bw*(5) format files into a *pix*(5) file. *Bw3-pix* reads the specified input files and writes stdout until all files have been exhausted. The first file goes into the red color plane, the second green, the third blue.

Bw3-pix accepts two special filename synonyms: "–", which stands for stdin, and ".", which stands for */dev/null*, the latter of which will write black into the corresponding color plane.

SEE ALSO

brlcad(1), pix-bw3(1), pix(5), bw(5)

AUTHORS

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwcrop – crop black and white bw files

SYNOPSIS

bwcrop infile outfile

DESCRIPTION

Bwcrop crops an arbitrarily large *bw*(5) input file (*infile*) to any size and writes the resulting output to *outfile*. *Bwcrop* prompts for all parameters it needs to do the cropping. All responses should be pixel counts or coordinates.

SEE ALSO

brlcad(1), bwrect(1), bwscale(1), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwdiff – compare two black and white files

SYNOPSIS

bwdiff [–{**mglen**}] file1 file2

DESCRIPTION

Bwdiff compares two black and white *bw*(5) format files in one of several ways, and writes the results on standard out The default method of comparison is to record the differences as offsets from neutral, half-intensity grey (128). The value written will be brighter than this neutral grey by an amount proportional to the signed difference between the pixels in *file1* and *file2*. That is, pixels which are brighter in *file1* than in *file2* will be brighter than the neutral grey, while pixels which are dimmer in *file1* than in *file2* will be dimmer. This difference is divided by two so that the results will fit in the 0->255 output range.

The **–m** flag takes the magnitude of the difference, thus no offset or divide is necessary. Brighter pixels indicate larger differences in value. Black indicates no difference in value.

The flags **–g** **–l** **–e** and **–n** are relational operations, standing for greater-than, less-than, equal-to, and not-equal-to respectively. Only one of these flags may be used at a time. In this mode the value of a pixel in the first file is compared to the corresponding pixel in the second, and if the relation is true, a full-white (255) pixel is output, else black. This mode is useful for producing masks showing pixels which satisfy the given condition.

Bwdiff accepts "–" as a synonym for stdin, and "." as a synonym for */dev/null*.

Note that this program also produces useful results on color *pix*(5) files.

EXAMPLE

Suppose you ran *pixstat*(1) on a file and found the modes to be red=107, green=93, blue=70. You could display those pixels whose colors are at these values by:

gencolor 107 93 70 bwdiff –e file.pix – pix-fb

SEE ALSO

brlcad(1), pixdiff(1), bwstat(1), bw(5), pix(5).

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwfilter – apply a 3x3 filter to a black and white bw file

SYNOPSIS

bwfilter [**–f** type] [**–v**] [**–d** div] [**–o** offset] [**–w** width] < file.bw > newfile.bw

DESCRIPTION

Bwfilter applies one of several 3x3 kernel filters to a black and white *bw*(5) format file. It will take a file name or else read from standard input. The input file size will be assumed to have a line length of 512 pixels unless the **–w** flag is used to tell it otherwise. By default a low pass filter is used. The **–f** *type* flag can be used to specify a different type of filter.

The *type* choices include:

lo low pass, the default.

vg vertical gradient

hg horizontal gradient

la laplacian

b boxcar low pass

Most filters divide the result of the kernel mutiplication by some value about in order to scale it back to a 0->255 range. For example, the boxcar divides by nine. The **-d** flag can be used to change this divisor. Some filters also add an offset to their results. The output of a laplacian for example is "centered" about zero, thus it adds 128 to the answer to yield a difference from neutral grey. The **–o** flag can bw used to change this offset.

The **–v** flag displays the kernel values on standard error.

BUGS

This should allow any 3x3 kernel to be specified.

SEE ALSO

brlcad(1), bwscale(1), bw(5).

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwhist – display a histogram of a black and white bw file on a framebuffer

SYNOPSIS

bwhist [**–v**] < file.bw

DESCRIPTION

Bwhist displays a histogram of the pixel values in a *bw*(5) format file. The output will be displayed on the frambuffer selected via the **FB_FILE** environment variable (see *libfb*(3)).

The relative frequency is displayed on the x axis scaled so that the maximum is full screen. The value 0 is at the bottom of the screen, 255 is at the top.
The output is overlayed on whatever is currently on the display.

The **–v** flag will output the histogram values on the terminal.

SEE ALSO

brlcad(1), bwhisteq(1), bwstat(1), bw(5)

BUGS

This program assumes it has at least 512 lines in y to work with.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwhisteq – equalize the histogram of a black and white bw file

SYNOPSIS

bwhisteq [**–v**] file.bw > file.equalized

DESCRIPTION

Bwhisteq automatically equalizes the histogram of a black and white *bw*(5) format file. It reads the file *file.bw* and writes a "remapped" version of it on it's standard output. The method used is to distribute the values of the histogram, as best as possible, in order to produce a constant average "density", while maintaining monotonicity.

The **–v** flag will output the old and new histogram values on the terminal.

SEE ALSO

brlcad(1), bwhist(1), bwstat(1), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwmod – arithmetically modify pixel values in black and white bw file

SYNOPSIS

bwmod {**–a add –s** sub **–m** mult **–d** div **–A –e** exp **–r** root} < file.bw

DESCRIPTION

Bwmod will perform a series of arithmetic operations on each 8-bit pixel of it's input/output stream. This is a *bw*(5) file format. The pixel values range from 0 to 255 and are internally operated on in floating point form so that dynamic range, truncation, and sign, are not a problem. On output they are converted back to 8-bit unsigned values clipping to 0 and 255 if necessary. The number of pixels clipped high and low, if any, are reported to standard error on completion.

Any number of operations can be given and they are applied in order. The choices are:
–a *val* to add a value.
–s *val* to subtract a value.
–m *val* to multiply by a value.
–d *val* to divide by a value.
–A to take to absolute value of each pixel.
–e *val* to exponentiate, i.e. raise each pixel to the power *val*.
–r *val* to take the *val* root of each pixel.

All arguments can be floating point values. Note that this command can be applied to color *pix*(5) files also, but no distinction can be made between color planes.

EXAMPLES

The command
bwmod –m–1 –a255 < file.bw > file2.bw
will produce the negative of a file by first mutiplying by minus one, and then adding 255 to shift the pixels back into the 0->255 range.

One can apply a gamma factor to a file as follows:
bwmod –d255 –r2.2 –m255 < file.bw > file2.bw
which will first normalize the pixel values, then take the 2.2 root of each pixel, and finally scale it back to 0->255. [Note however that applying gammas this way is not recommended since a function call per pixel results!]

Suppose you ran bwstat on a file and found min=80, max=210, mean=100. If you wish to expand the range of intensities, while still preserving the mean, this can be done by:
bwmod –s100 –m1.409 –a100 < file.bw > file2.bw
Where 1.409 results from (255-100)/(210-100) which is what one would use by boost the max value (210) to full scale (255) after subtracting the mean (100). Subtracting and adding 100 make the multiply occur "about that point", i.e value less than 100 will get smaller; those above, higher.

SEE ALSO

brlcad(1), pix(5), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwrect – extract a rectangle from a black and white bw file

SYNOPSIS

bwrect infile outfile

DESCRIPTION

Bwrect extracts a rectangular region from a black and white *bw*(5) format file. The program prompts for the area to extract in x and y pixels, the scanline length of the input file in pixels, and the origin in the input file from which to extract. First quadrant coordinates and file storage format are used.

SEE ALSO

brlcad(1), bwcrop(1), bwscale(1), pixrect(1), bw(5)

BUGS

We ought to allow the questions to be answered on the command line so that shell scripts, etc., could use this.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwrot – rotate, invert, or reverse a black and white bw file

SYNOPSIS

bwrot [–**rifb**] nx ny < file.bw > out.bw

DESCRIPTION

Bwrot will rotate, invert, or reverse a back and white *bw*(5) format file. The input file size in pixels of x and y is given.

The –**r** flag reverses the scan lines left to right.
The –**i** flag inverts the image top to bottom.
The –**f** and –**b** flag will rotate the image ninety degrees forward or backward respectively. The sense of the rotation is relative to a first quadrant coordinate system, i.e. origin lower left, where "forward" means clockwise.

Note that a 180 degree rotation can be obtained by both reversing and inverting.

METHOD

A fairly large buffer is kept in core so that, file size permitting, the entire rotation can occur without seeking on input or output. In any event, the algorithm is designed to process the input sequentially, so that if any seeking is required, it will happen on output only. If these seeks fail, e.g. the output is a pipe, the user will be told that it needs a file on output. A more general solution to this problem would be to see if seeks can be done on input instead, and switch the algorithm to seek on input only.

SEE ALSO

brlcad(1), pixrot(1), bwscale(5), bwrect(1), bw(5)

BUGS

If the file size is very large, excessive seeking may occur on output and the rotation will take a very long time. More complicated algorithms could remedy this.
Rotations of arbitrary angle are not supported.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwscale – change the size of a black and white bw file

SYNOPSIS

bwscale file.bw inx iny outx outy > newfile.bw

DESCRIPTION

Bwscale will take a black and white *bw*(5) format file, with dimensions *inx* by *iny* pixels, and produce a file with *outx* by *outy* pixels on it's standard output. The algorithm used is bilinear interpolation if scaling up, and a box filter of arbitrary size if scaling down. For the box filter a "square pixel" assumption is made; that is, all whole and fractional input pixels falling into a rectangle the size of an output pixel contribute equally according to the fraction of the output pixel area they cover. When interpolating, the edge pixels are preserved, i.e. all of the interpolated pixels fall inside of the edges. This can be a bit surprising when say doubling the size of a file as only the edge pixels will fall at exactly the location of an output pixel.

SEE ALSO

brlcad(1), bwcrop(1), bwrect(1), bw(5), pixscale(1)

BUGS

This program can not scale up in one dimension and down in the other at the same time.
The box filter used for scaling down results in the usual high-frequency ripple.
Optional cubic interpolation would be nice.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

bwstat – list statistics related to a black and white bw image.

SYNOPSIS

bwstat [**–v**] < file.bw

DESCRIPTION

Bwstat will take the given *bw*(5) format file, or read from standard in. A collection of statistics related to the pixel intensities is produced.

These include:

The number of pixels (and the square root of this which is the file dimensions if square).

The max and min values.

The mean.

The median (dividing line where half the pixels are above this value, half below).

The mode (most frequently occuring value) and the number of pixels at this value.

The variance, standard deviation, and skew.

The **–v** flag will also dump a histogram of the pixel values.

METHOD

Since the pixel values are quantized (have discrete values from 0 to 255), the statistics can be computed from "lump sums" after building a histogram of the input. This results in a fantastic savings in computation while also making a single pass over the input file sufficient.

SEE ALSO

brlcad(1), bwhist(1), pixstat(1), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

conv-vg2g – upgrade old to new format binary MGED database

SYNOPSIS

conv-vg2g file.vg file.g

DESCRIPTION

Conv-vg2g converts several older forms of binary *mged*(1) databases to the current form.

The conversion from version 3 to version 4 (the current version) is fairly straightforward. Conversion from older versions requires additional information, for which the user is prompted. The units of the database must be specified as: 1=mm, 2=cm, 3=meters, 4=inches, 5=feet. A title for the database is also requested.

SEE ALSO

brlcad(1), mged(1), libged(3)

DIAGNOSTICS

Error messages are intended to be self-explanatory, and usually indicate errors in the input database. It may be possible to rescue a damaged old database by suitably modifying the source code to the converter.

AUTHOR

Keith A. Applin

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

dunncolor – set exposure values for a Dunn Model 631 camera

SYNOPSIS

dunncolor [–**p**] baseval redval greenval blueval

DESCRIPTION

Dunncolor sets the exposure values in the Dunn Instruments Model 631 camera to the specified values. If the *-p* flag is used, the Polaroid (8X10) exposure values are set. Otherwise, the auxiliary camera exposure values are set. ***baseval, redval, greenval,*** and ***blueval,*** are values between 0 and 255 used to calculate the exposure times for the red, green, and blue filters. The final value for each color is the product of ***baseval*** and the value for that color (***redval***, ***greenval,*** or ***blueval***). Exposure timing is based on the counting of vertical SYNC pulses. The final exposure value indicates the number of SYNC pulses for which the camera's monitor should be unblanked. At a 60 Hz rate, this means an exposure time of 1 second for every 60 SYNC pulses. *Dunncolor* reads the old values from the camera and prints them before setting the new values. The new values are then read from the camera and printed to verify that they have been changed. The five values printed are the exposure values for base, red, green, blue and black/white.

EXAMPLES

35mm Kodachrome 64 at f/4, shaded images

dunncolor 2 80 16 16

35mm Kodachrome 64 at f/4, single-pixel wireframes, hires

dunncolor 14 70 16 18

16mm Video News Film 7250, f/8

dunncolor 1 56 14 14

Typically, red is 4X green and blue.

DIAGNOSTICS

Diagnostics are intended to be self-explanatory.

FILES

/dev/camera

SEE ALSO

brlcad(1), dunnsnap(1)

AUTHOR

Don Merritt

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

dunnsnap – expose film in a Dunn Model 631 camera

SYNOPSIS

dunnsnap [num_frames]

DESCRIPTION

Dunnsnap checks the status of the Dunn Instruments Model 631 camera and then attempts to take *num_frames* exposures. If *num_frames* is not supplied, the number of frames exposed defaults to one. Status of the camera is checked before every exposure. Limits on *num_frames* include your film budget and the mean time between failures of your computer, framebuffer and camera.

DIAGNOSTICS

Diagnostics are intended to be self-explanatory.

FILES

/dev/camera

SEE ALSO

brlcad(1), dunncolor(1)

AUTHOR

Don Merritt

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fb-bw – save a framebuffer image in a black and white bw file

SYNOPSIS

fb-bw [**–hi**] [**–X** scr_xoff] [**–Y** scr_yoff] [**–s** squaresize] [**–w** width] [**–n** height] [file.bw]

DESCRIPTION

Fb-bw will take a region of a framebuffer display and save it as a black and white *bw*(5) format file. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used. By default, the output file will be made size 512x512 pixels. Specifying the **–h** flag, changes the default to 1024x1024. The **–s** flag will set both the file height and width to be the given size, while **–w** and **–n** sets them individually.

A first quadrant coordinate system is used (origin in lower left of screen), and the area saved is normally from this origin. The **–X** and **–Y** flags can be used to move the origin to an offset position on the screen. The **–i** flag switches to a fourth quadrant system, both in the storage order as well as the origin offset coordinates (i.e. origin is now the upper left corner of the screen).

An equal blending of red, green, and blue is done. For more complicated black and white conversion use *fb-pix*(1) and *pix-bw*(1).

SEE ALSO

brlcad(1), bw-fb(1), fb-pix(1), pix-bw(1), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fb-pix – save a framebuffer image in a color pix file

SYNOPSIS

fb-pix [**–hi**] [**–s** squaresize] [**–w** width] [**–n** height] [file.pix]

DESCRIPTION

Fb-pix will take a region of a framebuffer display and save it as a color *pix*(5) format file. The environment variable **FB_FILE** is used to select the display device (see *brlcad(1)*). If this variable is not set, the default device for your system will be used. By default, the output file will be made size 512x512 pixels. Specifying the **–h** flag, changes the default to 1024x1024. The **–s** flag will set both the file height and width to be the given size, while **–w** and **–n** sets them individually.

A first quadrant coordinate system is used (origin in lower left of screen), and the area saved is normally from this origin. The **–i** flag switches to a fourth quadrant system, both in the storage order as well as the origin offset coordinates (i.e. origin is now the upper left corner of the screen).

SEE ALSO

brlcad(1), pix-fb(1), fb-bw(1), pix(5)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fb-rle – save a frame buffer image in run-length encoded form

SYNOPSIS

fb-rle [**–CShvw**] [**–c{rgbwBG}**] [**–l** x y] [**–p** x y] [file.rle]

DESCRIPTION

Fb-rle saves a frame buffer image in a file, using the University of Utah standard Run-Length Encoded (RLE) format. The environment variable **FB_FILE** is used to select the display device (see *brlcad(1)*). If this variable is not set, the default device for your system will be used. If *file.rle* is not specified, the encoded file is written to standard output. *Fb-rle* attempts to compress flat-colored areas as much as possible. This means that it is generally reasonable to save an entire image, since in RLE format it will not take up much more file space than extracting and saving only the interesting portion of the image. The default action is to encode only data pixels and to save the background color separately, so that when the image is restored all non-image pixels will be set to the background pixel value stored in the RLE header. Images saved in this mode may be displayed normally, or in "overlay" mode by the RLE decoder program (see *rle-fb* (1)). Alternatively, the **–S** option may be used to RLE encode the background as well (recommended only if you don't have a flat background).

–C Convert ("crunch") an image with a non-standard color map so that the RLE file created will redisplay with a standard color map. Useful to preserve dramatic images created with wierd color maps so that Gamma correction (for film) can then also be applied. This option is recommended when producing images for export to other display systems, so that variations in the interpretation of color maps between different types of display hardware do not affect the displayed image.

–w Convert the color image to black and white, and RLE encode that. The red, green, and blue components of each pixel are combined via the NTSC transform to get an intensity value (Y) which is saved. A more versatile tool for this is *pix-bw*(1).

–S Save all data in RLE format, irrespective of whether it is background or not ("straight box save"). It is not possible to restore a file saved with **–S** as an overlay.

–c*x* Specify a color to use as the "background". The character *x* is selected from the set **rgbwBG** (red, green, blue, white, Black, 18% Grey). If this option is not specified, the pixel in the lower left corner of the region being saved is used as the background color.

–l*x y* Specify the size of the saved box. This is useful if only a portion of the image is to be saved.

–p*x y* Specify the upper left corner of the saved portion of the image.

–v Verbose mode. The program will provide voluminous status information on its standard error as it proceeds through encoding the image.

–h "hires" flag. Set both the height and width of the framebuffer to 1024. The default is 512x512.

EXAMPLE

fb-rle –h tank.rle

AUTHORS

Gary S. Moss
Michael John Muuss

SEE ALSO

rle-fb(1), rle-pix(1), pix-rle(1), librle(3).

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbanim – animate small images on framebuffer using hardware pan and zoom

SYNOPSIS

fbanim [–**hirv**] [–**p** npasses] width nframes [fps]

DESCRIPTION

Fbanim assumes that the current framebuffer (determined by environment variable FB_FILE) contains a collection of small square images "tiled" together by *pixtile*(1). This permits a limited sequence of low resolution images to be viewed at various speeds, to create the sensation of motion, or to highlight the differences between the frames. The typical use of this program is to animate sequences of 64x64 or 128x128 tiles generated by animation studies. It can also be used to create "blink comparisons" between a small number of large images.

The width of each tile is specified by the *width* argument, and the number of tiles is specified by the *nframes* argument. The optional argument *fps* specifies the number of frames (tiles) per second to be displayed. 24 fps is the standard movie rate, 30 the standard video rate. Slower rates may also be used. The default is 8 frames/second.

By default, the framebuffer is assumed to be loaded with a 512x512 pixel image. Specifying the –**h** flag, changes the default to 1024x1024.

By default, the sequence of tiles is considered to be a circular (repeating) sequence, ie, the first tile immediately follows the last one. If this is not the case, the –**r** flag will cause the sequence to "rock" from beginning to end to beginning, which is sometimes more pleasing.

The –**v** flag will cause the frame number and center coordinates of each frame to be printed as they are passed to fb_window(), for debugging purposes.

By default, the *fbanim* program repeats for 100 passes. When creating demonstration scripts, it may be desirable to specify the number of passes through the animation. The –**p** *npasses* option is used to indicate this.

For compatability with old animation images which were generated by fourth-quadrant programs, where the first image starts in the upper left corner of the screen rather than the lower left corner, the –**i** flag is used to indicate this inverse order.

SEE ALSO

brlcad(1), pixtile(1), pix-fb(1), pix(5)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbbackgnd – display a shaded background on a framebuffer

SYNOPSIS

fbbackgnd [**–h**] [**–t** title_height] hue saturation

DESCRIPTION

Fbbackgnd displays a shaded background on a framebuffer suitable for making slides for a presentation on. The upper portion of the screen is reserved for a title area, and is drawn at constant intensity 240, followed by a 4 scanline white bar, after which the intensity smoothly decreases from 240 to 50. *Hue* and *saturation* are values from 0 to 255 and can be selected via *fbcolor*(1). A hue of 160 is a pleasing blue.

By default, the title height is 80 scanlines in lowres, and 90 scanlines in hires, but the **–t** *title_height* option may be used to specify another value. If *title_height* is set to zero, then the white bar is omitted as well, and the entire display is smoothly shaded.

By default, the display is assumed to be 512x512 pixels; by specifying the **–h** flag, the output size is changed to 1024x1024.

EXAMPLES

fbbackgnd 160 200
fbbackgnd -t 120 160 250

SEE ALSO

brlcad(1), fbcolor(1).

AUTHOR

Michael John Muuss

BUGS

The starting and ending intensities should probably be an option. This program should probably generate a *pix*(5) format file as output, instead of writing to a framebuffer.

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbclear – clear a frame buffer image

SYNOPSIS

fbclear [–**h**] [red green blue]

DESCRIPTION

Fbclear is used to erase a frame buffer using whatever method is efficient for the device being accessed. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used. If the –**h** option is used on those devices where it makes a difference (e.g. Adage), the **HIRES** (1024x1024x24) memory is cleared; otherwise a **LORES** (512x512x24) device is assumed. If the optional color values are given, *fbclear* writes the specified color over the entire image. The color values must be in the range 0 to 255.

SEE ALSO

brlcad(1), fbcmap(1)

AUTHORS

Gary S. Moss
Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbcmap – change the color map of a frame buffer

SYNOPSIS

fbcmap [**–h**] [map_number]

DESCRIPTION

Fbcmap is used to change the color map of a frame buffer. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used. If the **–h** option is used, a **HIRES** (1024x1024x24) framebuffer is written; otherwise a **LORES** (512x512x24) device is assumed. This size distinction is important only for devices like the Adage framebuffers which have different operating modes based on the display size.

If an optional color *map_number* is given, that map is used; the default is the standard linear map. There are currently six other maps supplied numbered 1 through 6 as follows:

1 reverse-linear (negative) map.

2 one corrected for Polaroid 809/891 film.

3 a linear mapping in which the first 100 values are black.

4 a linear map with values 1->191 incremented by 64 to boost dim pictures. The values at 191 and above are all full on. Zero remains at zero.

5 a gamma correcting color map used by the University of Utah Alpha_1 project.

6 a map with a series of color deltas. This map is mostly black with the following colors, appearing at multiples of 32, having values full on: white (0), magenta (32), red (64), yellow (96), green (128), cyan (160), blue (192). This map is most useful in conjunction with *fbcmrot*(1) and will show which pixels have the same intensities in one, two, or all three color planes.

There are also three special maps used to quickly change the display to a solid color for adjusting a camera without losing image memory, numbered 10 through 12 for black, white, and 18% neutral grey, respectively.

SEE ALSO

brlcad(1), fbcmrot(1).

AUTHORS

Gary S. Moss
Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbcmrot – rotate the color map

SYNOPSIS

fbcmrot [**–h**] steps_per_second

DESCRIPTION

Fbcmrot rotates the color map, cycling each color map entry through all of the color map positions. The *frames_per_second* is the number of shifts per second to generate. This value can be fractional. A *frames_per_second* of zero or an unspecified frame rate causes the program to cycle through the color maps as fast as possible.

By default, the framebuffer is assumed to be 512x512 pixels. Specifying the **–h** flag, changes this to 1024x1024. This is only important for devices such as the Adage framebuffers which operate differently depending on the display size.

SEE ALSO

brlcad(1), fbcmap(1)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbcolor – interactively pick colors

SYNOPSIS

fbcolor [**–h**]

DESCRIPTION

Fbcolor is an interactive color picking facility. It allows you to pick and display colors, by varying parameters in either RGB (red, green, blue) or HSI (hue, saturation, intensity) space. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used.

Fbcolor draws a set of color bars in the lower left corner of the display, sets the remainder of the display to the current color, then starts reading standard in for color editing commands. As you change the various parameters which specify the color, the color bars grow and shrink to show the relative red, green and blue intensities.

At the same time, *fbcolor* displays the current RGB and HSI values on your terminal. This allows you to record the RGB or HSI values of the colors you select for later use.

Fbcolor recognizes two main sets of commands; those which select which parameter to edit, and those which vary the value of the selected parameter. "**r**", "**g**", and "**b**" selects the red, green, or blue color for editing, while "**h**", "**s**", and "**i**", (or "**v**") selects the hue, saturation, or intensity for editing. There is one other parameter selection character, the "/", which steps through the list of color parameters in the order given above.

"**+**" and "**.**" both increment the currently selected parameter by one; "**-**" and "**,**" both decrement by one, and "**>**" and "**<**" increments and decrements by 16. The value of each of the parameters can range from 0 to 255 and all parameters wrap in both directions.

A "**?**" prints a short command summary, and typing "**q**", "**Q**", "**<CR>**", or "**<LF>**" exits the program.

There is also a **–h** flag, which uses the frame buffer in high resolution mode.

SEE ALSO

brlcad(1), fbclear(1), fbcmap(1)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbed – framebuffer editor

SYNOPSIS

fbed [**–ph**]

DESCRIPTION

Fbed is a screen editor for framebuffer images.

If the **–p** option is supplied, *fbed* uses the GTCO bitpad to position the cursor on the screen. This option does not prevent the user from positioning the cursor with the keyboard commands also. The **–h** option is for editing 1024x1024 images; the default is 512x512.

HINTS

This program may require a little practice, be sure to save a copy of the input files until you are confident. There is an on-line menu available by typing a '?'. The first time using the program you should start out by listing the menu and finding the command for saving (Run-Length Encoding) your image and use it if you don't already have a copy.

fbed will work on UNIX files if you don't need to see what is happening.

CAVIATS

As far as data tablets go, there is currently only support for the GTCO bitpad and it assumes that there is only one available. The subroutines dealing with this should be made into a library and generalized to allow for different hardware and multiple devices on the system.

WARNING

Be careful not to use the **–p** option unless you are sitting next to a GTCO bitpad which is operational, or the program will hang.

EXAMPLE

fbed –h

SEE ALSO

fb-rle(1), rle-fb(1), libfb(3)

AUTHOR

Gary S. Moss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbframe – place a colored frame around a framebuffer

SYNOPSIS

fbframe [**–h**]

DESCRIPTION

Fbframe places a colored frame around the edge pixels of a framebuffer. It is mostly for purposes of framebuffer alignment. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If the variable is not set, the default device for your system will be used. If the **–h** option is used, a 1024x1024 frame is written; otherwise a 512x512 device is assumed.

The frame starts in the lower left with a red line running to the right to the next to last pixel. Next, a green line from there up to the next to top scan line is drawn. Then blue across the top scan line to the next to first column. Finally a white line down the left edge stopping just above the red line.

SEE ALSO

brlcad(1), fbgrid(1)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbgrid – draw a grid on a frame buffer

SYNOPSIS

fbgrid [–**hcbdo**]

DESCRIPTION

Fbgrid is used to generate a grid on a frame buffer. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used. If the –**h** option is used, a 1024x1024 grid is written; otherwise a 512x512 device is assumed.

–**c** will clear the screen before drawing.

The available patterns are:

–**b** for binary spacing with lines at 8, 64, and 128 line intervals.

–**d** for decimal spacing with lines at 10, 50, and 100 line intervals (the default).

–**o** for the old cross grid pattern.

SEE ALSO

brlcad(1), fbpoint(1), fbzoom(1), fbframe(1)

AUTHORS

Phillip Dykstra
Gary S. Moss
Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbpoint – point at and identify screen coordinates

SYNOPSIS

fbpoint [**–h**] [**–x**[prefix]] [**–y**[prefix]]

DESCRIPTION

Fbpoint allows you to move a non-destructive cursor around on the display. As the cursor is moved around, *fbpoint* writes the coordinates of the pixel pointed at by the cursor, and the red, green, and blue values of that pixel, on standard error *fbpoint* also writes the last set of coordinates pointed to on standard out on exit.

The "**h**", "**j**", "**k**", and "**l**", commands step the cursor one pixel left, down, up, or right respectively, and the "**H**", "**J**", "**K**", and "**L**", commands step the cursor in the corresponding direction in big increments. The "**B**", "**N**", "**P**". and "**L**", and "**^B**", (control-B), "**^N**", "**^P**", and "**^F**", are also little and big (respectively) cursor movement commands.

A "**?**" prints a short command summary, and typing "**q**", or "**Q**", exits the program.

The **–h** flag causes the frame buffer to be used in high resolution mode (1024x1024). This is important for frame buffers such as the Adage which operate differently depending on the display size. Without this flag the default size for the selected device will be used (usually the entire screen).

Using this program with grave accents in a shell script in order to interactively grab a pair of coordinates is one of it's main intentions. This is the reason for the feedback on standard error and the final coordinate pair on standard out. By default, both the x and y coordinates are written. If the **–x** or **–y** flags are given only those coordinates will be written. In these cases an optional *prefix* can follow the flag which will be appended to the output coordinate. This is to facilitate passing the output of this program as arguments to another.

EXAMPLE

fbpoint
by itself is used to pick coordinates or pixel values from a display.

fbscanplot 'fbpoint –y'
allows you to choose the scanline to plot using fbpoint.

pix-fb 'fbpoint –x–X –y–Y' file.pix
will allow you to select an x, y screen coordinate and then place the lower left corner of the image *file.pix* at that location.

SEE ALSO

brlcad(1), fbgrid(1), fbzoom(1)

AUTHORS

Michael John Muuss
J. Robert Suckling

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbscanplot – plot the intensities of a framebuffer scanline

SYNOPSIS

fbscanplot [**–h**] [**–v**] [**–o**] [**–r**] [**–W** scr_width] [**–F** outframebuffer] yline

DESCRIPTION

Fbscanplot will plot the red, green, and blue, pixel intensities of a selected scanline of a framebuffer. The line to be plotted has a first quadrant y coordinate of *yline*, i.e. a value of zero indicates the bottom line of the display. The environment variable **FB_FILE** is used to select the input display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used. The output will appear on this same device unless a **–F** *outframebuffer* flag is given to redirect the output.

If no size if specified the selected device(s) will be opened at it's default size (usually the full screen). The flag **–W** *scr_width* can be used to set a screen width (scanline length) other than this default. The **–h** flag is a shorthand for a size of 1024x1024.

Either the bottom 256 or top 256 scanlines will be used to output the plot; whichever will not cover the selected line. A black background is drawn with horizontal grey lines at intervals of 16 and 64 scanlines in order to provide a scale. The intensities of the selected line are then plotted, the bottom of the graph being intensity zero; the top 255.

The **–o** flag will plot the scanline in overlay mode. With this flag no background or grid lines are drawn, and the output always appears in the bottom 256 scanlines of the display. Using this option more of the image will remain visible, and multiple scanlines can be plotted on top of each other for comparison. The image may make the plot harder to see however.

If the **–r** flag is given, a reverse image of the selected scanline is drawn, i.e. the intensities from 0->255 become 255->0. This is simply to highlight the line selected. The plot will still be of the non-reversed values.

The **–v** flag causes the pixel values to be output to the terminal as well as displayed.

EXAMPLE

fbscanplot -r 37
will plot the intensities of the scanline 37 up from the bottom. This line will be hightlighted by replacing it with its negative. The output will appear in the top 256 scanlines so as not to cover the selected line.

fbscanplot –v –W512 –o 'fbpoint –y' > scan.out
will use *fbpoint* to select a y scanline, then plot 512 pixel values in overlay mode. These values will also be saved in the file *scan.out* for closer examination.

SEE ALSO

brlcad(1), fbgrid(1), fbzoom(1)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

fbzoom – enlarge a frame buffer image

SYNOPSIS

fbzoom [**–h**]

DESCRIPTION

Fbzoom is used to zoom-in on a frame buffer image by enlarging each pixel and windowing in on a portion of the image. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used.

The "**h**", "**j**", "**k**", and "**l**", commands step the center of view one pixel left, down, up, or right respectively, and the "**H**", "**J**", "**K**", and "**L**", commands step the center of view in the corresponding direction in big increments. The "**B**", "**N**", "**P**". and "**L**", and "^**B**", (control-B), "^**N**", "^**P**", and "^**F**", are also little and big (respectively) center of view movement commands.

A "**?**" prints a short command summary, and typing "**q**", or "**Q**", exits the program.

The **–h** flag causes the frame buffer to be used in high resolution mode (1024x1024). This is important for frame buffers such as the Adage which operate differently depending on the display size. Without this flag the default size for the selected device will be used (usually the entire screen).

SEE ALSO

brlcad(1)

AUTHORS

Gary S. Moss
Michael John Muuss
J. Robert Suckling

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

g2asc, asc2g – convert between binary and ASCII MGED databases.

SYNOPSIS

g2asc < file.g > file.asc
asc2g < file.asc > file.g

DESCRIPTION

The local machine binary representation of an *mged*(1) database is inherently non-portable. *G2asc* and *asc2g* are simple filters which convert between the binary form and a portable ASCII representation. Any warning messages are written to standard error.

G2asc and *asc2g* are typically used to convert binary *mged*(1) files to a portable ASCII format so the files can be transferred to another machine. On the local machine, files are converted to ASCII using *g2asc*(1), then transferred to the remote machine. On the remote machine, the files are then converted back to the binary representation for that machine using *asc2g*(1).

EXAMPLE

g2asc < file.g > file.asc
rcp file.asc remote:file.asc
rsh remote "asc2g < file.asc > file.g"

DIAGNOSTICS

The diagnostics are intended to be self-explanatory.

SEE ALSO

brlcad(1), mged(1)

AUTHOR

Charles Michael Kennedy
Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

gencolor – output a pattern of byte values

SYNOPSIS

gencolor [–**r** num] [val1 ... valN]

DESCRIPTION

Gencolor repeats a pattern of up to 8192 8-bit values on it's standard output. The values, which range from 0 to 255, are taken either from the command line, or in binary from it's standard input. The **–r** flag can be used to specify the number of times to repeat the given pattern, the default being infinity. This program is useful for producing a given grey value or color.

EXAMPLE

gencolor -r512 0 255 0 > foo.pix
Will output 512 "green" pixels

SEE ALSO

brlcad(1), pix(5), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

loop – generate integer indices for shell loops, etc

SYNOPSIS

loop start finish [increment]

DESCRIPTION

Loop generates a sequence of integers between ***start*** and ***finish*** inclusive, one per line, on stdout. This is mostly useful for driving loops when writing Shell programs *sh*(1). The default increment is +1 when start<finish, and -1 when start>finish. An optional ***increment*** may be specified. If ***increment*** is zero, then the ***start*** value will be generated continuously.

EXAMPLE

```
for i in 'loop 10 30 5'
do
  echo $i
done
```

Will display the values 10, 15, 20, 25, and 30.

BUGS

This program just scratches the surface. The shells could benefit from a greater ability to perform math. Limitless opportunities for extensions exist, but this simple version is adequate for most shell scripts dealing with sequences of images, etc.

AUTHORS

Phillip Dykstra
Douglas P. Kingston III
Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

mged – multi-display interactive combinatorial solid geometry editor

SYNOPSIS

mged database.g

DESCRIPTION

Mged interactively edits the specified database file *database.g* on any one of a number of display devices, including any UNIX-plot device, a Vector General 3300, a Megatek 7250, a Tektronix 4014 or compatable display, a Tektronix 4109 or compatable display, a Raster Technologies One/80, an Evans and Sutherland Picture System 300, or a Silicon Graphics IRIS 2400 or 3030 workstation.

For detailed documentation, obtain a copy of the guide "Computer Graphics for Target Descriptions" or BRL Technical Report ARBRL-TR-02480, "GED: An Interactive Solid Modeling System for Vulnerability Assessments".

SIGINT aborts the current command.

"q" or EOF exits the program.

HINTS

Mged database names conventionally end with **.g**.

EXAMPLE

$ ***mged tank.g***

SEE ALSO

Computer Graphics for Target Descriptions
rt(1), libged(3), librt(3).

DIAGNOSTICS

Error messages are intended to be self-explanatory.

AUTHORS

Keith Applin
Doug Gwyn
Chuck Kennedy
Gary Kuehl
Gary Moss
Mike Muuss
Chuck Stanley
Bob Suckling
Earl Weaver

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pix-bw – combine pix color planes into a bw black and white image

SYNOPSIS

pix-bw [**–ntsc**] [**–crt**] [**–R**[num]] [**–G**[num]] [**–B**[num]] < file.pix > file.bw

DESCRIPTION

Pix-bw will combine or select the color planes of a ***pix***(5) format file from it's standard input and produce a black and white *bw*(5) file on it's standard output. By default all three colors are weighted equally. The **–ntsc** uses weighting based on the NTSC primaries and the D6500 alignment white. The **–crt** flag applies weights for "typical" CRT phosphor chromaticities and the D6500 white.

Individual color plane(s) can be selected via the **–R –G** and **–B** flags. If one or more of these flags is followed by a value, the value will be treated as a weight to be applied to that color This value is implicitly 1.0 divided by the number of colors selected for the flags which are not followed by a value.

EXAMPLES

pix-bw –R –B < file.pix > file.bw
will equally blend the red and blue color planes.

pix-bw –R0.30 –G0.59 –B0.11 < file.pix > file.bw
has the same effect as the **–ntsc** flag.
[The **–crt** weights are: 0.26, 0.66, 0.08]

SEE ALSO

brlcad(1), pix-bw3(1), pix(5), bw(5).

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pix-bw3 – split up the color planes of a color pix file

SYNOPSIS

pix-bw3 redout greenout blueout < file.pix

DESCRIPTION

Pix-bw3 reads a color *pix*(5) format file and outputs the red, green, and blue color planes into the given files. These files will then be in *bw*(5) format.

SEE ALSO

brlcad(1), bw3-pix(1), pix(5), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pix-fb − display a pix format RGB image file on a framebuffer

SYNOPSIS

pix-fb [−**hicz**] [−**F** framebuffer] [−**s** squarefilesize] [−**w** file_width] [−**n** file_height] [−**S** squarescrsize] [−**W** scr_width] [−**N** scr_height] [−**x** file_xoff] [−**y** file_yoff] [−**X** scr_xoff] [−**Y** scr_yoff] < file.pix

DESCRIPTION

Pix-fb reads a *pix*(5) format file from the named file, or from standard in if no file is specified, and displays the image on the currently selected framebuffer. The environment variable FB_FILE specifies the current framebuffer, see *brlcad*(1). Alternatively, the framebuffer may be explicitly specified by using the −**F** flag.

By default, the *pix* file, and the requested framebuffer size, is assumed to be 512x512 pixels. Specifying the −**h** flag changes the size to 1024x1024.

If the −**i** flag is specified, the image is output from top to bottom. Customarily, *pix* files are stored bottom to top, so this flag inverts the image.

The −**c** flag causes the screen to be cleared before the image is displayed. The default behavior is to overwrite the current image with the new image without clearing the screen.

The −**z** flag will zoom and center on the image being displayed to make it fill the display area.

The −**w** *file_width* flag specifies the width of each scanline in the input file, in pixels. −**W** *scr_width* does the same for the display device.

The −**n** *file_height* and −**N** *scr_height* flags specifies the height in scanlines of the input file or display device respectively.

−**s** *squarefilesize* and −**S** *squarescrsize* set both the height and width to the size given.

−**x** *file_xoff* −**y** *file_yoff* and −**X** *scr_xoff* −**Y** *scr_yoff* will offset into the file or onto the screen by the given amounts. The coordinate system for these offsets is first quadrant, with the origin at the lower left corner of the image.

FORMAT

A *pix*(5) file contains sequences of pixels. Each pixel is stored as three **unsigned chars**, first red, then green, then blue. The first pixel in a *pix* file is the lower left corner of the image. The pixels proceed from left-to-right across each scanline, with scanlines being written from the bottom to the top of the image. By convention *pix* images are usually square.

SEE ALSO

rt(1), fb-pix(1), libfb(3), pix(5)

DIAGNOSTICS

If the *pix* file is shorter than expected, the program exits silently.

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pix-rle – convert a pix format RGB file into run-length encoded form

SYNOPSIS

pix-rle [**–hdv**] [**–s** squarefilesize] [**–w** file_width] [**–n** file_height]
< file.pix > file.rle

DESCRIPTION

Pix-rle saves a *pix*(5) format image in a Run-Length Encoded (RLE) form. If *file.pix* is not specified, the *pix*(5) file is read from standard in. If *file.rle* is not specified, the encoded file is written to standard out. *Pix-rle* attempts to compress flat-colored areas as much as possible. This means that it is generally reasonable to save an entire image, since in RLE format it will not take up much more file space than extracting and saving only the interesting portion of the image.

–w *file_width*
: Specify the width (in pixels) of the scanlines in the input file. The default is 512.

–n *file_height*
: Specify the number of scanlines of input to be encoded. The default is 512.

–s *squarefilesize*
: Specify both the number of pixels per scanline and the number of scanlines in the file to process.

–h
: "hires" flag. Set both the height and width of the input to 1024. The default is 512x512.

–v
: Verbose mode. The program will provide voluminous status information on standard error as it proceeds through encoding the image.

–d
: Debug mode.

EXAMPLE

pix-rle <plane.pix plane.rle
pix-rle –h car.pix car.rle
pixfilter -f lo <truck.pix pix-rle > truck-filt.rle

AUTHOR

Michael John Muuss

SEE ALSO

brlcad(1), rle-pix(1), pix-fb(1), fb-pix(1), librle(3).

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pix2asc, asc2pix – encode and decode color pixel files

SYNOPSIS

pix2asc < file.pix > file.a_pix
asc2pix < file.a_pix > file.pix

DESCRIPTION

Pix2asc and *asc2pix* are simple filters which convert color *pix*(5) files from their binary form to a portable ASCII form, and back. *Pix2asc* reads the binary file on standard in and writes ASCII on standard out. *Asc2pix* does the reverse.

The binary file consists of a sequence of three-tuples of **unsigned chars** representing the red, green, and blue intensities for each pixel. The ASCII version of the file has one RGB pixel per newline-terminated line, with each pixel represented as six hexadecimal digits.

When moving *pix*(5) files between UNIX systems via tape (see *tar*(1) and *cpio*(1)) or via network (see *rcp*(1) and *ftp*(1)), this conversion to ASCII is unnecessary, as all UNIX files are byte streams, regardless of the underlying hardware. However, when transporting images to non-UNIX machines, these tools are invaluable. Note that *rt*(1) can directly generate the ASCII form of an image file using the **–O** flag, allowing it to be easily used on non-UNIX systems.

SEE ALSO

brlcad(1), pix(5)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixbustup – break up concatenated pix images into seperate files

SYNOPSIS

pixbustup basename width [image_offset] [first_number] < input.pix

DESCRIPTION

Pixbustup takes input consisting of a concatenated collection of *pix*(5) format files and breaks them up into seperate files. The output file names will be of the form *basename.num*, where *basename* is given on the command line, and *num* is a consecutive series of integers. By default the first image will be made number " 0 ", but an alternate *first_number* can be given. The images must be square and the width (and height) is given by *width*. The program keeps reading these small files until end of file is reached. If one does not wish to extract from the beginning of the input file an *image_offset* can be given which will skip that number of images before beginning.

This program is mostly a holdover from the days when *rt*(1) wrote animations into one huge file, but it is still sometimes useful.

SEE ALSO

brlcad(1), rt(1), pix(5)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixdiff – highlight pixel colors which differ between two files

SYNOPSIS

pixdiff file1.pix file2.pix > out.pix

DESCRIPTION

Pixdiff highlights the pixels which differ between two color *pix*(5) format files. The input files are read pixel by pixel and if any of the color planes red, green, or blue differ, those colors are displayed at full intensity. If none of the color values differ, an NTSC converted black and white equivalent at one half the original intensity is displayed. This is to aid in seeing where in the image the different pixels lie.

Also see *bwdiff*(1) which can still be used on *pix* files, yet can produce different kinds of output.

SEE ALSO

brlcad(1), bwdiff(1), pix-fb(1), pixstat(1), pix(5)

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixfilter – apply a 3x3 filter to a color pix file

SYNOPSIS

pixfilter [**–f** type] [**–v**] [**–d** div] [**–o** offset] [**–w** width] < file.pix > newfile.pix

DESCRIPTION

Pixfilter applies one of several 3x3 kernel filters to a color *pix*(5) format file. It will take a file name or else read from standard input. The input file size will be assumed to have a line length of 512 pixels unless the **–w** flag is used to tell it otherwise. By default a low pass filter is used. The **–f** *type* flag can be used to specify a different type of filter.

The *type* choices include:

lo low pass, the default.

vg vertical gradient

hg horizontal gradient

la laplacian

b boxcar low pass

Most filters divide the result of the kernel mutiplication by some value in order to scale it back to a 0->255 range. For example, the boxcar divides by nine. The **-d** flag can be used to change this divisor. Some filters also add an offset to their results. The output of a laplacian for example is "centered" about zero, thus it adds 128 to the answer to yield a difference from neutral grey. The **–o** flag can be used to change this offset.

The **–v** flag displays the kernel values on standard error.

SEE ALSO

brlcad(1), pixscale(1), pix(5)

BUGS

This should allow any 3x3 kernel to be specified.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixhist – display a histogram of a color pix file on a framebuffer

SYNOPSIS

pixhist [**–v**] < file.pix

DESCRIPTION

Pixhist displays a histogram of the pixel values in a *pix*(5) format file. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used.

The relative frequency is displayed on the x axis scaled so that the maximum is full screen. The value 0 is at the bottom of the screen, 255 is at the top. All three colors will be drawn on each scan line, and all are scaled by the same factor.
The output is overlayed on whatever is currently on the display.

The **–v** flag will also output the histogram values on the terminal.

SEE ALSO

brlcad(1), pixhist3d(1), pixstat(1), pix(5)

BUGS

This program assumes it has at least 512 lines in y to work with.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixhist3d-pl – output RGB color space utilization in 3D unix-plot form

SYNOPSIS

pixhist3d-pl [file.pix]

DESCRIPTION

Pixhist3d-pl takes a color *pix*(5) format file and outputs points for each RGB color occuring in the image. The format of these points is the BRL extended three dimensional UNIX plot (see *libplot3*(3)). A red, green, and blue line is drawn along the edges of the color space indicating the coordinate axis. The resolution of the color space is 128x128x128, i.e. the least significant bit of each color plane is ignored.

These plots are best viewed on an interactive 3D UNIX plot display program, so that the color cube can be viewed from all angles. Anomalies in the colors of an image can often be seen, e.g. oversaturation resulting in intensity clipping.

SEE ALSO

brlcad(1), pixhist3d(1), pixhist(1), libplot3(3), plot-fb(1), pix(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixhist3d – display RGB color space utilization

SYNOPSIS

pixhist3d < file.pix

DESCRIPTION

Pixhist3d builds up a count of color frequencies in the RGB color space and then displays three orthographic views of this color space. An RGB color cube with red, green, and blue values from 0 to 255 form a right handed coordinate system. Three faces of this cube are displayed where the intensity of the pixels shown for each face correspond the the integrated frequencies through the color cube perpendicular to that face. These frequencies are scaled so that the maximum value is full white, and any non-zero scaled count less than 30 is displayed at intensity 30 so as to be visible.

The faces are:
Lower left: red vertical, green horizontal
Upper left: blue vertical, green horizontal
Lower right: red vertical, blue horizontal

The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used.

SEE ALSO

brlcad(1), pixhist3d-pl(1), pixhist(1), pixstat(1) pix(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixinterp2x – interpolate a color pix file twice as large as the original

SYNOPSIS

pixinterp2x [**–h**] [**–s** squarefilesize] [**–w** file_width] [**–n** file_height] < infile.pix

DESCRIPTION

Pixinterp2x does a bilinear interpolation between thc pixels of the input *pix*(5) file to create the output file. This program is most useful when creating images for photographing. Even though no additional image information is created, there may be significant photographic advantages to operating the display at it's highest resolution.

This is a very fast implementation of one special case of interpolation; the general tool for interpolation is *pixscale*(1).

–h "hires" flag. Set both the height and width of the input to 1024. The default is 512x512.

–w *file_width*
Specify the width (in pixels) of the scanlines in the input file. The default is 512.

–n *file_height*
Specify the number of scanlines of input to be processed. The default is 512.

–s *squarefilesize*
Specify both the number of pixels per scanline and the number of scanlines in the file to process.

The interpolated scanlines fall exactly half way between the input lines, and the input lines are exactly preserved. Rather than extrapolate beyond the edges of the file, the first X and Y line is duplicated (left and bottom edge). This differs from the behavior of *pixscale*(1).

SEE ALSO

brlcad(1), pixscale(1), pix(5).

BUGS

More memory is used than is strictly necessary.

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixrect – extract a rectangle from a color pix file

SYNOPSIS

pixrect [–#bytes] infile outfile

DESCRIPTION

Pixrect extracts a rectangular region from a color *pix*(5) format file. The program prompts for the area to extract in x and y pixels, the scanline length of the input file in pixels, and the origin in the input file from which to extract. First quadrant coordinates and file storage formats are used.

By default pixels of size three bytes are assumed as in the *pix*(5) format. However, the *–#bytes* flag can be used to specify some other pixel size, where *bytes* is the number of bytes per pixel. Thus this program can be used to extract rectangles of other data types, such as an array of floats, etc.

SEE ALSO

brlcad(1), bwrect(1), pix(5), bw(5)

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixrot – rotate, invert, or reverse a color pix file

SYNOPSIS

pixrot [[–#bytes] [–**rifb**] nx ny < file.pix > newfile.pix

DESCRIPTION

Pixrot will rotate, invert, or reverse a color *pix*(5) format file. *Nx* and *ny* specify the input file x and y sizes in pixels.

The **–r** flag reverses the scan lines left to right.
The **–i** flag inverts the image top to bottom.
The **–f** and **–b** flag will rotate the image ninety degrees forward or backward respectively. The sense of the rotation is relative to a first quadrant coordinate system, i.e. origin lower left, where "forward" means clockwise.

Note that a 180 degree rotation can be obtained by both reversing and inverting.

By default three byte pixels are assumed as in the *pix*(5) format. However, the *–#bytes* flag can be used to specify some other pixel size, where *bytes* is the number of bytes per pixel. Thus this program can be used to manipulate rectangles of other data types, such as an array of floating point numbers, files of text, etc.

METHOD

A fairly large buffer is kept in core so that, file size permitting, the entire rotation can occur without seeking on input or output. In any event, the algorithm is designed to process the input sequentially, so that if any seeking is required, it will happen on output only. If these seeks fail, e.g. the output is a pipe, the user will be told that it needs a file on output. A more general solution to this problem would be to see if seeks can be done on input instead, and switch the algorithm to seek on input only.

SEE ALSO

brlcad(1), pixscale(1), pixrect(1), pix(5).

BUGS

If the file size is very large, excessive seeking may occur on output and the rotation will take a very long time. More complicated algorithms could remedy this.
Rotations of arbitrary angle are not supported.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixscale – change the size of a color pix file

SYNOPSIS

pixscale file.pix inx iny outx outy > newfile.pix

DESCRIPTION

Pixscale will take a color *pix*(5) format file with dimensions *inx* by *iny* pixels, and produce a file with *outx* by *outy* pixels on stdout. The algorithm used is bilinear interpolation if scaling up, and a box filter of arbitrary size if scaling down. For the box filter a "square pixel" assumption is made; that is, all whole and fractional input pixels falling into a rectangle the size of an output pixel contribute equally according to the fraction of the output pixel area they cover. When interpolating, the edge pixels are preserved, i.e. all of the interpolated pixels fall inside of the edges. This can be a bit surprising when say doubling the size of a file as only the edge pixels will fall at exactly the location of an output pixel.

For 2X interpolation, a much faster alternative is *pixinterp2x*(1).

SEE ALSO

brlcad(1), pixcrop(1), pixrect(1), pixinterp2x(1), pix(5).

BUGS

This program can not scale up in one dimension and down in the other at the same time.
The box filter used for scaling down results in the usual high-frequency ripple.
Optional cubic interpolation would be nice.

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixstat – list statistics related to a color pix image

SYNOPSIS

pixstat [**–v**] < file.pix

DESCRIPTION

Pixstat will take the given *pix*(5) format file, or read from standard in. A collection of statistics related to the pixel intensities is produced for each color plane (red, green, and blue).
These include:
The number of pixels (and the square root of this which is the file dimensions if square).
The max and min values.
The mean.
The median (dividing line where half the pixels are above this value, half below).
The mode (most frequently occuring value) and, with the **–v** flag, the number of pixels at this value.
The variance, standard deviation, and skew.

The **–v** flag will also dump a histogram of the pixel values.

METHOD

Since the pixel values are quantized (each color having discrete values from 0 to 255), the statistics can be computed from "lump sums" after building a histogram of the input. This results in a fantastic savings in computation while also making a single pass over the input file sufficient.

SEE ALSO

brlcad(1), pixhist(1), bwstat(1), pix(5).

AUTHOR

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pixtile – paste small images together into one large image

SYNOPSIS

pixtile [**–h**] [**–s** squareinsize] [**–w** file_width] [**–n** file_height] [**–S** squareoutsize] [**–W** out_width] [**–N** out_height] [**–o** startframe] basename > file.pix

pixtile [**–h**] [**–s** squareinsize] [**–w** file_width] [**–n** file_height] [**–S** squareoutsize] [**–W** out_width] [**–N** out_height] file1 file2 ... fileN > file.pix

DESCRIPTION

Pixtile takes a collection of numbered small images and places them side by side in one larger image. The input and output files are assumed to be in *pix*(5) format. By default, the output file will be made 512x512 pixels in size. Specifying the **–h** flag, changes the default size to 1024x1024 pixels.

The input files have a default width of 64 pixels per scanline and are usually square, i.e. have the same height as width. Specifying the **–s** flag, changes the default size to *squareinsize* pixels per scanline and *squareinsize* scanlines as given in the argument list. The **–w** flag can be used to specify the input file width in pixels and the **–n** flag can be used to specify the input file height in scanlines.

If the **–S** flag is supplied, the output size is changed from the default value of 512 to *squareoutsize* pixels per scanline and *squareoutsize* scanlines. The **–W** flag sets the output width size to *out_width* pixels per scanline. Similarly, the **–N** flag sets the output height size to *out_height* scanlines.

–h "hires" flag. Set both the height and width of the output to 1024. The default is 512x512.

–s *squareinsize*
Specify both the number of pixels per scanline and the number of scanlines in the input files. The default is 64.

–w *file_width*
Specify the width (in pixels) per scanline of the input file. The default is 64.

–n *file_height*
Specify the number of scanlines to be read from the input file. The default is 64.

–S *squareoutsize*
Specify both the number of pixels per scanline (width) and the number of scanlines (height) for the output file. The default is 512.

–W *out_width*
Specify the width (in pixels) per scanline of the output file. The default is 512.

–N *out_height*
Specify the number of scanlines to be sent to the output file. The default is 512.

–o *startframe*
Specify the starting frame number, where *startframe* is substituted for *num* in *basename.num*

The files being pasted together should have names of the form *basename.num*, where *basename* is given on the command line, and *num* is a consecutive series of integers. By default the first image is assumed to be number " 0 ", but an alternate *startframe* can be given by using the **–o** flag and supplying *startframe*. The program keeps reading these small files until either the next frame does not exist, or the output file is full. They are placed in the output file from lower left to upper right. The progress of the composition is displayed on standard error.

Alternatively, if more than one *basename* (filename) argument is supplied on the command line, the names of the files being pasted together are assumed to be explicitly stated and the **-o** flag is not valid. This allows files with different names to be used without having to change the name of the files.

This program is the companion of *fbanim*(1) which will step through these small images on a framebuffer capable of zooming and windowing. Thus many small low resolution frames can be used as an animation "preview" for a higher resolution sequence, the latter of which would go onto video tape or film.

EXAMPLE

pixtile -h -s 128 -o 2 anim > out.pix
will read files *anim.2* through *anim.65* placing them from lower left to upper right in a 1024x1024 pixel file *out.pix*.

pixtile -s 256 lowerleft lowerright upperleft upperright > out.pix
will read files *lowerleft*, *lowerright*, *upperleft*, and *upperright* placing them from lower left to upper right in a 512x512 file *out.pix*.

SEE ALSO

brlcad(1), fbanim(1), pix(5).

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

plot-fb – display 3-D UNIX plot on a framebuffer

SYNOPSIS

plot-fb [**–hoid**] [filename]

DESCRIPTION

Plot-fb displays a BRL extended three dimensional UNIX plot file on a framebuffer. The environment variable **FB_FILE** is used to select the display device (see *brlcad(1)*). If this variable is not set, the default device for your system will be used. By default, the display size is assumed to be 512x512 pixels. Specifying the **–h** flag, changes the output size to 1024x1024.

The **–o** flag is for overlay mode, where the current framebuffer image will be left in the background. The **–i** flag causes the rasterized image to be plotted immediately, rather than buffering up rasterized data into bands. While useful for quick feedback on small plots, this mode is generally more expenisve in CPU time. The **–d** flag outputs a very verbose debugging stream to standard error listing every UNIX plot command and the corresponding framebuffer coordinates. If *filename* is not specified, the plot commands are read from standard in.

Note that 3-D UNIX plot is a superset of the standard UNIX plot format, thus this program can display normal plot files as well.

SEE ALSO

brlcad(1), libplot3(3).

AUTHORS

Douglas A. Gwyn
Michael John Muuss
Joseph C. Pistritto

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

pp-fb – display a GIFT pretty-picture file on a frame buffer

SYNOPSIS

pp-fb file.pp

DESCRIPTION

Pp-fb reads *file.pp* , determines the size, and displays a color shaded image represented therein on a frame buffer. The *.pp* files are the output of the pretty-picture option of the *gift* (1V) ray-tracing program. Some information pertaining to the color assignment of region identification numbers and the background color is prompted for on the standard input. The environment variable **FB_FILE** is used to select the display device (see *brlcad*(1)). If this variable is not set, the default device for your system will be used. Commands for manipulating the display are documented on-line. The file format is not documented anywhere.

SEE ALSO

gift(1V), brlcad(1).

AUTHORS

Gary Kuehl
Michael John Muuss
Gary S. Moss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

rle-fb – decode run-length encoded image onto a frame buffer

SYNOPSIS

rle-fb [**−Otv**] [**−b** { **rgbBG** }] [**−p** x y] [file.rle]

DESCRIPTION

Rle-fb decodes a University of Utah standard Run-Length Encoded (RLE) format file (saved with *fb-rle* (1) or a similar program) to a frame buffer device or a file. If *file.rle* is not specified, standard input is read. *Rle-fb* determines the type and size of the saved file (color or B&W) and acts accordingly. The environment variable **FB_FILE** is used to select the display device (see *brlcad(1)*). If this variable is not set, the default device for your system will be used.

−t Alters the normal bottom-up display order to be top-down. This option can be useful when the image contains accompaning text. Note that the image is inverted in memory, using as large a buffer as *malloc(3)* will permit. This means that the entire image will be decoded before output will begin appearing.

−v Provides a verbose description of the parameters of the saved image on standard error.

−O Restores the image data in *overlay* mode. Only areas of the original image which were not the background color are decoded. The rest of the image already in the frame buffer is undisturbed. Note that this mode uses twice as many DMAs to the frame buffer as the normal mode, as the previous image must be read back into the host.

−b*x* If the image is not being decoded in overlay mode, the background is normally set to the value recorded in the RLE file. This option permits the user to specify an alternate background color from a limited set of colors. The character *x* can be any of **r** (red), **g** (green), **b** (blue), **B** (black), or **G** (18% grey).

−p*x y* Reposition the image. The original lower left corner is positioned at (*x* , *y*) before restoring the image. Warning: a saved image should not be repositioned so that any saved data wraps around the X borders. If the input image is 512x512, and it is repositioned, the framebuffer will be opened in 1024x1024 mode.

file.rle Name of file to decode, in RLE format.

EXAMPLES

rle-fb truck.rle
rle-fb −Ot text.rle
rle-fb −bG <car.rle

SEE ALSO

brlcad(1), fb-rle(1), rle-pix(1), pix-rle(1), librle(3).

AUTHORS

Gary S. Moss
Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

rle-pix – restore run-length encoded image and convert to RGB pixel format

SYNOPSIS

rle-pix [**–v**] [**–b** { **rgbBG** }] [file.rle]

DESCRIPTION

Rle-fb decodes a University of Utah standard Run-Length Encoded (RLE) format file (saved with *fb-rle* (1), *pix-rle* (1), or a similar program), converts it into *pix* (5) format, and writes it to standard out. If *file.rle* is not specified, standard input is read. *Rle-pix* determines the type and size of the saved file (color or B&W) and acts accordingly. If a non-linear colormap is stored in the RLE file, is is applied to the pixels as they are output.

–v Provides a verbose description of the parameters of the saved image on standard error.

–b*x* The background of the image is normally set to the value recorded in the RLE file. This option permits the user to specify an alternate background color from a limited set of colors. The character *x* can be any of **r** (red), **g** (green), **b** (blue), **B** (black), or **G** (18% grey).

file.rle Name of file to decode, in RLE format.

EXAMPLE

rle-pix tank.rle > tank.pix

SEE ALSO

brlcad(1), fb-rle(1), pix-fb(1), pix(5).

AUTHOR

Gary S. Moss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

rt – raytrace an **mged** model, giving a shaded picture

SYNOPSIS

rt [options ...] model.g objects ...

DESCRIPTION

Rt operates on the indicated *objects* in the input *model.g* and produces a color shaded image as the output. By default, the output is placed on the current framebuffer as it is computed. The environment variable FB_FILE specifies the current framebuffer, see *brlcad*(1). Alternatively, the output can be stored in a pixel file (*pix*(5) format) by specifying **–o output.pix** in the option list. The orientation of the rays to be fired may be specified by the **–a** and **–e** options, in which case the model will be autosized, and the grid will be centered on the centroid of the model, with ray spacing chosen to span the entire set of *objects*. Alternatively, with the **–M** option, a transformation matrix may be provided on standard input which maps model-space to view-space. In this case, the grid ranges from -1.0 <= X,Y <= +1.0 in view space, with the size of the grid (number of rays fired) specified with **–f .** The **–M** option is most useful when **rt** is being invoked from a shell script created by an *mged*(1) *saveview* command.

The following options are recognized.

–f# Number of rays to fire in X and Y directions (square grid). Default is 512 (512x512), maximum is 1024 (1024x1024).

–a# Select azimuth in degrees. Used with **–e** and conflicts with **–M**

–e# Select elevation in degrees. Used with **–a** and conflicts with **–M**

–M Read model2view matrix from standard input. Conflicts with **–a** and **–e**

–o output.pix Specify the name of a file to store the *pix*(5) format output. The *pix-fb*(1) utility can be used to later display ".pix" files.

–O output.a_pix Specify the name of a file to store the ASCII version of the image. *asc2pix*(1) can be used to later convert the file to *pix*(5) format output.

–x# Set debug flags to (hexadecimal) number. Not generally useful.

–l# Select lighting model. Default is 0. Model 0 is a full lighting model with the ability to implement Phong shading, transparant and reflective objects, shadow penumbras, texture maps, etc. In addition to ambient light, a small amount of light is supplied from eye position. If an object called "LIGHT" exists in the active model space, then it provides a second light source, and shadow computations will be initiated automaticly. (This mechanism is subject to change). Model 1 is a diffuse-lighting model only, and is intended for debugging. Model 2 displays the surface normals as colors, and is useful for examining curvature and surface orientation. Model 3 is a three-light diffuse-lighting model, and is intended for debugging.

The **rt** program is a simple front-end to *librt*(3) which can be invoked directly, or via the **rt** command in *mged*(1).

SEE ALSO

mged(1), rtray(1), pix-fb(1), rtray(1), rtpp(1), librt(3), ray(5V), pix(5).

DIAGNOSTICS

Numerous error conditions are possible. Descriptive messages are printed on standard error.

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

COPYRIGHT

This software is Copyright (C) 1985 by the United States Army. All rights reserved.

BUGS

Most deficiencies observed while using the **rt** program are usually with the *librt*(3) package instead.

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

rtpp – raytrace an **mged** model, writing a GIFT-style Pretty Picture file

SYNOPSIS

rtpp [options ...] model.g objects ... > model.pp

DESCRIPTION

Rtpp operates on the indicated *objects* in the input *model.g* and produces a GIFT-style "pretty picture" format file on standard output. The orientation of the rays to be fired may be specified by the **–a** and **–e** options, in which case the model will be autosized, and the grid will be centered on the centroid of the model, with ray spacing chosen to span the entire set of *objects*. Alternatively, with the **–M** option, a transformation matrix may be provided on standard input which maps model-space to view-space. In this case, the grid ranges from -1.0 <= X,Y <= +1.0 in view space, with the size of the grid (number of pixels) specified with **–f** This option is most useful when **rtpp** is being invoked from a shell script created by an *mged*(1) *saveview* command.

The following options are recognized.

–f# Number of pixels in X and Y directions (square grid). Default is 512 (512x512), maximum is 1024 (1024x1024).

–a# Select azimuth in degrees. Used with **–e** and conflicts With **–M**

–e# Select elevation in degrees. Used with **–a** and conflicts with **–M**

–M Read model2view matrix from standard input. Conflicts with **–a** and **–e**

–x# Set debug flags to (hexadecimal) number. Not generally useful.

The **rtpp** program is a simple front-end to *rtlib*(3) which is most useful when used with *mged*(1). In order to display a pretty-picture file, the programs *pp-ik*(1) and *pp-fb*(1) are used.

SEE ALSO

gift(1V), mged(1), pp-ik(1), pp-fb(1), rt(1), rayplot(1), rayvect(1), rtpp(1), rtray(1), rtlib(3), ray(5V).

DIAGNOSTICS

Numerous error conditions are possible. Descriptive messages are printed on standard error (file descriptor 2).

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

COPYRIGHT

This software is Copyright (C) 1985 by the United States Army. All rights reserved.

BUGS

Most deficiencies observed while using the **rtpp** program are usually with the *rtlib*(3) package instead.

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

rtray – raytrace an **mged** model, writing a VLD rayfile

SYNOPSIS

rtray [options ...] model.g objects ... > model.ray

DESCRIPTION

Rtray operates on the indicated *objects* in the input *model.g* and produces a VLD standard *ray*(5V) format ray file on standard output. The orientation of the rays to be fired may be specified by the **–a** and **–e** options, in which case the model will be autosized, and the grid will be centered on the centroid of the model, with ray spacing chosen to span the entire set of *objects*. Alternatively, with the **–M** option, a transformation matrix may be provided on standard input which maps model-space to view-space. In this case, the grid ranges from -1.0 <= X,Y <= +1.0 in view space, with the size of the grid (number of rays fired) specified with **–f** This option is most useful when **rtray** is being invoked from a shell script created by an *mged*(1) *saveview* command.

The following options are recognized.

–f# Number of rays to fire in X and Y directions (square grid). Default is 512 (512x512), maximum is 1024 (1024x1024).

–a# Select azimuth in degrees. Used with **–e** and conflicts With **–M**

–e# Select elevation in degrees. Used with **–a** and conflicts with **–M**

–M Read model2view matrix from standard input. Conflicts with **–a** and **–e**

–x# Set debug flags to (hexadecimal) number. Not generally useful.

The **rtray** program is a simple front-end to *rtlib*(3) which is most useful when used with *mged*(1) and the VLD ray filters. In order to convert a ray file into a minimal monochrome UNIX-plot image for viewing, run

```
/vld/bin/rayvect -mMM < file.ray > file.vect
/vld/bin/vectplot -mMM < file.vect > file.plot
tplot -Tmeg file.plot          # or equivalent UNIX-plot filter
```

SEE ALSO

gift(1V), mged(1), rt(1), rayplot(1), rayvect(1), rtray(1), rtpp(1), rtlib(3), ray(5V).

DIAGNOSTICS

Numerous error conditions are possible. Descriptive messages are printed on standard error (file descriptor 2).

AUTHOR

Michael John Muuss

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

COPYRIGHT

This software is Copyright (C) 1985 by the United States Army. All rights reserved.

BUGS

The upper limit of 1024x1024 is pointless, and should be removed.

Most deficiencies observed while using the **rtray** program are usually with the *rtlib*(3) package instead.

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

libfb – multiple device, generic frame buffer library

SYNOPSIS

```
#include <fb.h>

/* Generic frame buffer routines: */

FBIO *fb_open(fbfile, width, height)
char *fbfile;

int fb_close(fbp)
FBIO *fbp;

int fb_read(fbp, x, y, addr, count)
FBIO *fbp;
RGBpixel *addr;
long count;

int fb_write(fbp, x, y, addr, count)
FBIO *fbp;
RGBpixel *addr;
long count;

int fb_rmap(fbp, cmap)
FBIO *fbp;
ColorMap *cmap;

int fb_wmap(fbp, cmap)
FBIO *fbp;
ColorMap *cmap;

int fb_clear(fbp, colorp)
FBIO *fbp;
RGBpixel *colorp;

char *fb_gettype(fbp)
FBIO *fbp;

int fb_getwidth(fbp)
FBIO *fbp;

int fb_getheight(fbp)
FBIO *fbp;

/* Hardware specific frame buffer routines: */

int fb_cursor(fbp, mode, x, y)
FBIO *fbp;

int fb_scursor(fbp, mode, x, y)
FBIO *fbp;

int fb_setcursor(fbp, bits, xbits, ybits, xorig, yorig)
FBIO *fbp;
unsigned char bits[];
int xbits, ybits;
int xorig, yorig;

int fb_window(fbp, x, y)
FBIO *fbp;
```

```
int fb_zoom(fbp, x, y)
FBIO *fbp;

/* Buffered frame buffer I/O: */

void fb_ioinit(fbp)
FBIO *fbp;

int fb_seek(fbp, x, y)
FBIO *fbp;

void fb_tell(fbp, xp, yp)
FBIO *fbp;
int *xp, *yp;

int fb_rpixel(fbp, pixelp)
FBIO *fbp;
RGBpixel *pixelp;

int fb_wpixel(fbp, pixelp)
FBIO *fbp;
RGBpixel *pixelp;

int fb_flush(fbp)
FBIO *fbp;

void fb_log(format [, arg ] ... )
char *format;
```

DESCRIPTION

These routines are designed to provide a device-independent method of using frame buffers or files containing frame buffer images. The coordinate system used is first-quadrant (0..width-1, 0..height-1), with integer addressing. Translation to hardware coordinate systems is handled by the library.

This version of the library assumes that red, green, and blue intensities are described by unsigned 8-bit bytes in the range (0..255).

The exact interpretation of color maps tends to be somewhat device specific. The three ColorMap arrays each have 256 entries of unsigned 16-bit values. In order to accomodate color maps with differing amounts of output resolution, the color map entries are fixed-point fractions in the range (0.0..1.0). In integer notation, the range is (0..65525). For devices with less than 16 bits of output from their color maps, the left-most portion of each entry is used.

Fb_open is used to open a frame buffer file *fbfile* . The file may be either the name of a supported frame buffer interface, referenced as "/dev/interface", or the name of a UNIX file. The routine will try to determine if the file opened was a real frame buffer by examining the name, and if so will perform whatever initialization actions are necessary. If the value of *fbfile* is NULL and the environment variable FB_FILE is set, then the value of FB_FILE is used; otherwise a default frame buffer device is used. A "/dev/debug" interface exists which just logs all calls to the library. If the remote interface is available, remote devices can be accessed as "hostname:/dev/device". The *width* and *height* parameters specify the initial size of display desired. If these are zero the default sizes for that device will be used. On a successful open, the frame buffer I/O (FBIO) structure pointer is returned. This structure contains size you were actually given, as well as the maximum possible size for the selected device. A return of FBIO_NULL indicates failure.

Fb_close simply closes the frame buffer.

Fb_read reads *count* pixels from the frame buffer starting at the location specified by x and y , and places them at program memory address specified by *addr* .

Fb_write writes *count* pixels from program address *addr* into the frame buffer starting at the location specified by *x* and *y* . *Fb_read* and *Fb_write* both return the number of pixels actually read or written, or -1 on error.

Fb_rmap reads in the color map from the frame buffer and leaves at the location pointed to by *cmap* .

Fb_wmap writes the color map pointed to by *cmap* into the frame buffer. If the value of *cmap* is NULL then a linear color map is used as the default.

Fb_clear erases the frame buffer by setting all pixels to the given color. If the color pointer is NULL, black will be used. On a UNIX file, this entails writing the entire file, which is an expensive operation, whereas on most frame buffer displays this can be done in less than a second by a special command.

Fb_gettype returns a pointer to a string describing the frame buffer specified by the FBIO pointer.

Fb_getwidth and *Fb_getheight* returns the current size of the FBIO frame buffer.

The following routines work in conjunction with those described above to provide functions which only apply if the frame buffer file is actually a hardware frame buffer display.

Fb_cursor places the cursor at the image space coordinates given by *x* and *y* . If the mode is non-zero, the cursor is made visible, and if mode is zero, the cursor is turned off.

Fb_scursor is the same as *fb_cursor* except that it places the cursor at the **screen** space coordinates given by *x* and *y* .

Fb_setcursor allows the user to set the bitmap used to represent the cursor, thereby changing the cursor shape. This is not necessarily supported by all hardware. The argument *bits* is a pointer to an array of unsigned chars containing the bits of the cursor. The arguments *xbits* and *ybits* specify the size of the cursor bitmap. The number of bytes in the *bits* array will be the width rounded up to a mutiple of eight (so that the cursor "scanlines" are byte aligned) times the height. *bits*[0] is the lower left corner, *bits*[1] is to the right of it, etc. The next line of the *bits* array goes above the current one. Within a byte the most significant bit is the leftmost. The values *xorig* and *yorig* specify which bit in the bitmap actually gets placed at the location specified in the cursor move routines. Again, a first quadrant coordinate system is used.

Fb_window sets the frame buffer window center position to the image space coordinates given by *x* and *y* . This command is usually used in conjunction with the *fb_zoom* routine.

Fb_zoom sets the zoom factor for the X coordinate to *x* and the zoom factor for the Y coordinate to *y* . Zooming is generally done by pixel replication in hardware.

The following routines work in conjunction with those described above to provide buffered reading and writing of frame buffer images either to a real frame buffer or a UNIX file. The routines use a simple paging strategy to hold "bands" of the image in core. Since horizontal bands are buffered, the ideal motion is to scan left to right, then bottom to top.

Fb_iobinit should be called before using any of the other buffered I/O routines and repeated whenever the frame buffer is reopened.

Fb_seek is used to position the current read/write pointer to the location to the next position to be read or written. It is not necessary to do a *fb_seek* after every read or write since both *fb_rpixel* and *fb_wpixel* imply an automatic move to the next pixel. If you read or write the last pixel on a scan line, the pointer will automatically move to the beginning of the following scan line.

Fb_tell returns the current location of the read write pointer in terms of (X,Y) coordinates on the frame buffer. The X and Y values are returned into the integers pointed to by *xp* and *yp* .

Fb_rpixel reads the pixel at the current frame buffer location and returns it into the location specifed by *pixelp* .

Fb_wpixel writes the pixel pointed to by *pixelp* at the current frame buffer location.

Fb_flush caused any current buffered frame buffer pages to be written out. Unnecessary writes are avoided by the use of page reference bits.

The following is a printing routine which this library uses to indicate errors.

Fb_log will convert, format and print its *args* under control of *format* to the standard error output. For more detailed information on the specification of the control string, see *printf* (3S). This function may be supplied by the application if different behavior is desired.

DEVICES

The following devices are supported by the library; not all may be available on any given system. New device support can be incorporated by the addition of a single module to the library.

/dev/debug*[num]*
: Debug interface
num is a bitvector indicating the levels of verbosity of the output. See **fb.h** for the bit definitions.

filename
: Disk file interface

/dev/ik*[num]*
: Adage RDS-3000 ("Ikonas")
Num indicates the device number.

/dev/sgi*[num]*
: Silicon Graphics IRIS Workstation
A non-zero *num* will select the pre-allocated colormap. The default is to dynamically allocate the colors of the colormap, which is slower but usually of higher quality. Note that both forms of colormapping occur only in window mode, not RGB mode.

/dev/rat
: RasterTek 1/80 (untested)

hostname:*devicename*
: TCP-based network links to a remote framebuffer, where *devicename* is any from this list.

EXAMPLE

Libfb can be loaded with any C program:

```
$ /bin/cc program.c -lfb
```

FILES

fb.h
/usr/brl/lib/libfb.a

SEE ALSO

brlcad(1).

RETURN VALUES

fb_close , *fb_write* , *fb_read* , *fb_wmap* , *fb_rmap* , *fb_clear* , *fb_cursor* , *fb_scursor* , *fb_setcursor* , *fb_window* , *fb_zoom* , *fb_seek* , *fb_wpixel* , *fb_rpixel* and *fb_flush* return −1 to indicate failure. *Fb_open* returns FBIO_NULL to indicate failure, and a non-null FBIO structure pointer upon success. *fb_read* , and *fb_write* return the number of pixels actually read or written. *fb_gettype* returns a pointer to NULL terminated a description string.

AUTHORS

Gary S. Moss
Phillip Dykstra
Mike J. Muuss
Douglas P. Kingston III

BUGS

Vertical scanning will incur the most overhead, making it almost impractical. Due to the way memory is organized in frame buffers and UNIX files, vertical scanning will never be easy unless the image can be rotated.

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

libplot3 – graphics interface subroutines

SYNOPSIS

```
pl_space( fp, x0, y0, x1, y1 )
FILE *fp;
int x0, y0, x1, y1;

pl_3space( fp, x0, y0, z0, x1, y1, z1 )
FILE *fp;
int x0, y0, z0, x1, y1, z1;

pl_erase( fp )
FILE *fp;

pl_color( fp, r, g, b )
FILE *fp;
int r, g, b;

pl_linmod( fp, s )
FILE *fp;
char *s;

pl_label( fp, s )
FILE *fp;
char *s;

pl_line( fp, x1, y1, x2, y2 )
FILE *fp;
int x1, y1, x2, y2;

pl_3line( fp, x1, y1, z1, x2, y2, z2 )
FILE *fp;
int x1, y1, z1, x2, y2, z2;

pl_move( fp, x, y )
FILE *fp;
int x, y;

pl_3move( fp, x, y, z )
FILE *fp;
int x, y, z;

pl_cont( fp, x, y )
FILE *fp;
int x, y;

pl_3cont( fp, x, y, z )
FILE *fp;
int x, y, z;

pl_point( fp, x, y )
FILE *fp;
int x, y;

pl_3point( fp, x, y, z )
FILE *fp;
int x, y, z;

pl_box( fp, x1, y1, x2, y2 )
FILE *fp;
```

```
int x1, y1, x2, y2;

pl_3box( fp, x1, y1, z1, x2, y2, z2 )
FILE *fp;
int x1, y1, z1, x2, y2, z2;

pl_circle( fp, x, y, r )
FILE *fp;
int x, y, r;

pl_arc( fp, x, y, x0, y0, x1, y1 )
FILE *fp;
int x, y, x0, y0, x1, y1;
```

DESCRIPTION

These subroutines generate graphic output commands for processing with the **plot(1)** plotting filters. They are slightly more general than those in **libplot** as these take a file pointer. They also include the BRL 3-D extensions to the plot intermediate code.

Pl_space or *pl_3space* must be used before any of the graphic primitives to declare the amount of space necessary. See *plot3(5)*.

Pl_box or *pl_3box* draws a box between the two given opposite points. The "pen" will be left at the second point.

Pl_circle draws a circle of radius *r* with center at the point (*x*, *y*). Note that *circle* and *arc* can not be transformed in three space if one is using a filter to do that.

Pl_arc draws an arc of a circle with center at the point (*x*, *y*) between the points (*x0*, *y0*) and (*x1*, *y1*).

String arguments to *pl_label* and *pl_linmod* are terminated by nulls and do not contain new-lines.

See *plot3(5)* and *plot(5)* for a description of the effect of the remaining functions.

FILES

/usr/lib/libplot3.a produces output for *plot* (1G) filters

WARNINGS

In order to compile a program containing these functions in *file.c* it is necessary to use "cc *file.c* –lplot3".

Color specification and three-dimensional primitives are BRL extensions to the "unix plot" language that are not generally found on other systems.

SEE ALSO

plot(1), plot-fb(1), plot3(5), plot(5)

AUTHORS

Phillip Dykstra

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

librle – run-length encoded image library

SYNOPSIS

```
#include <fb.h>
#include <rle.h>

int rle_rhdr( stream, flags, bgpixel )
FILE *stream;
int *flags;
Pixel *bgpixel;

rle_whdr( stream, ncolors, bgflag, cmflag, bgpixel )
FILE *stream;
Pixel *bgpixel;

rle_rlen( xlen, ylen )
int *xlen, *ylen;

rle_rpos( xpos, ypos )
int *xpos, *ypos;

rle_wlen( xlen, ylen, mode )
int xlen, ylen, mode;

rle_wpos( xpos, ypos, mode )
int xpos, ypos, mode;

int rle_rmap( stream, cmap )
FILE *stream;
ColorMap *cmap;

rle_wmap( stream, cmap )
FILE *stream;
ColorMap *cmap;

rle_decode_ln( stream, scanline_buffer )
FILE *stream;
Pixel *scanline_buffer;

rle_encode_ln( stream, scanline_buffer )
FILE *stream;
Pixel *scanline_buffer;
```

DESCRIPTION

These routines are designed to provide a means of decoding or encoding University of Utah Alpha_1 format Run-Length Encoded (RLE) files from a C program. There are separate routines for reading or writing the RLE header, reading or writing the encoded color map, and decoding or encoding a scanline of a raster image. The program that loads this library can be ignorant of the particular encoding scheme being used. This library does not need to know anything about the available frame buffer hardware. The frame buffer library, *libfb* (3), can also be used to free programs of hardware dependence.

The following routines are used to decode an RLE file into a raster image:

Rle_rhdr will seek to the beginning of the given input *stream* and read the setup information in the RLE header. This routine must be called first when decoding an RLE file. If *bgpixel* is not NULL, the background color stored in the RLE header will be placed there. *Flags* is a flag word using a combination of single-bit flags NO_BOX_SAVE, NO_COLORMAP, and NO_IMAGE (described in **rle.h**) to indicate whether the image was saved with background, whether the color map was saved, and whether the image was saved or just the color map. If the color map was

saved, *rle_rmap* must be used next to read the color map from the RLE file and decode it into the buffer pointed to by *cmap* . *Rle_rlen* and *rle_rpos* are used after *rle_rhdr* to retrieve, from the header structure, the length and position of the image in the locations specified by *xlen* , *ylen* , *xpos* and *ypos* .

Rle_decode_ln is used to decode the next scanline (starting at the bottom) and should be passed the address of a *scanline_buffer* large enough to store *xlen* plus *xpos* pixels. Only foreground pixels are written; it is the caller's responsibility to fill the buffer first with the appropriate background. If no foreground pixels occur, zero is returned; otherwise 1 is returned to indicate that the buffer has been altered.

The following routines are used to compress a raster image into an RLE file:

Rle_whdr is used first to specify the encoding scheme to use and to write the RLE header which will contain this information to the given output *stream* . The *ncolors* parameter specifies the number of color channels. Normally, this should be set to **3** (for red, green, and blue channels). If *ncolors* equals **1**, a monochrome (black and white) image is assumed, and if it is **0**, only the color map will be saved. *Bgflag* specifies whether or not the image is saved with the background. If *bgflag* is nonzero, only the foreground image is saved. If *bgflag* is zero, the entire image is encoded. *Rle_whdr* stores the background color along with the other setup information. If a monochrome image has been specified, the NTSC standard is used to calculate the background color from *bgpixel* ; otherwise *bgpixel* is used as the background color. *Cmflag* is nonzero to indicate that the color map is to be saved, and this is accomplished with *rle_wmap* which must be used next to encode the color map pointed to by *cmap* and write it to the RLE file. If the value of *cmap* is **NULL**, a standard (linear) color map will be written.

Rle_encode_ln is used to encode the next scanline (starting at the bottom) and should be passed the address of a *scanline_buffer* containing the pixels to be converted.

The *Pixel* and *ColorMap* data types are defined in **fb.h**. The color map and scanlines can be conveniently read from or written to a file or supported frame buffer with compatible functions found in *libfb* (3).

WARNINGS

The current version of this library reads Edition-1 and Edition-2 RLE files. The University of Utah is currently preparing Edition-3 of this format. All encoding is in Edition-2 format. Note that Edition-1 formats are byte-order sensitive, so that they can not be transported around the network with *rcp*(1) with abandon.

The RLE files must be decoded seqentially from the bottom up, as it is the convention that the origin be the lower left corner of the screen. *Rle_decode_ln* should fail if called more than *ylen* times, as it will attempt to read past the end of the input file. Encoding an RLE file from the top down is considered anti-social behavior and will lead to confusion if the installed frame buffer utilities are used.

The library functions must be used *in sequence* to read/write the header, followed by the color map (if one exists) and then the image. It is safe to process multiple images as long as the proper sequence is followed. Both *rle_rhdr* and *rle_whdr* seek to the beginning of *stream* before performing any I/O, but all other functions rely on sequential read/write operations on the RLE file.

Processing two images asynchronously is not recommended; that is, you should finish one before you begin the next. The problem stems from the fact that some global information is stored by *rle_rhdr* , and this information is clobbered by subsequent calls. Therefore, when switching back to an image that is partially decoded, the file offset must be determined with *ftell* (3S); then *rle_rhdr* can be invoked again to set up things, and finally *fseek* (3S) can be used to reposition the file pointer.

EXAMPLE

The library may be loaded as follows:

cc program.c −lrle.a

SEE ALSO

rle-fb(1), fb-rle(1), libfb(3).

DIAGNOSTICS

Upon error, all functions print a message and return −1.

AUTHORS

Gary S. Moss, BRL/VLD-VMB
Mike J. Muuss, BRL/SECAD
Spencer W. Thomas, U. of Utah

BUGS

It would be nice to fix the problem of asynchronous calls by passing a pointer to storage for the setup structure to the read/write header and read/write scanline routines so that there is no global information to contend with.

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

NAME

librt – library for raytracing an MGED database

SYNOPSIS

```
#include "raytrace.h"
#include "vmath.h"

extern struct rt_functab rt_functab[];
extern struct rt_g rt_g;

struct rt_i *rt_dirbuild( mged_file_name, buf, len )
char *mged_file_name;
char *buf;
int len;

int rt_gettree( rtip, object_name )
struct rt_i *rtip;
char *object_name;

int rt_shootray( ap )
struct application *ap;

void rt_prep_timer()

double rt_read_timer( buf, len )
char *buf;
int len;

void rt_pr_partitions( phead, title )
struct partition *phead;
char *title;

void rt_pr_seg( segp );
struct seg *segp;

void rt_bomb( msg )
char *msg;

void rt_log( msg )
char *msg;

void rt_viewbounds( )

struct soltab *rt_find_solid( )

void mat_zero( m )          /* fill matrix m with zeros */
matp_t m;

void mat_idn( m )           /* fill matrix m with identity matrix */
matp_t m;

void mat_copy( o, i )       /* copy matrix i to matrix o */
matp_t o, i;
```

```
void mat_mul( o, i1, i2 )       /* multiply i1 by i2 and store in o */
matp_t o, i1, i2;

void matXvec( ov, m, iv )       /* multiply m by vector iv, store in ov */
matp_t ov, m, iv;

void mat_inv( o, i )            /* invert matrix i, store result in o */
matp_t o, i;

void mat_print( title, m )      /* print matrix m, (with title) on stdout */
char *title;
matp_t m;

void mat_trn( o, i )            /* transpose matrix i into matrix o */
matp_t o, i;

void mat_ae( o, a, e )          /* rotation matrix o from azimuth+elevation */
matp_t o;
double a, e;

void mat_angles( o, a, b, g )   /* rotation matrix o from angles a, b, g */
matp_t o;
double a, b, g;

void vtoh_move( v, h )                  /* homogeneous vector from ordinary vector */
vectp_t v, h;

void htov_move( h, v )                  /* ordinary vector from homogeneous vector */
vectp_t h, v;
```

DESCRIPTION

rt_dirbuild opens *mged_file_name* and builds a directory for quick lookup of objects. *rt_dirbuild* returns a pointer to a struct rt_i on success (often called "rtip"), or RTL_NULL on failure (such as being unable to open the named database). This rt_i pointer must be saved, as it is a required parameter to *rt_gettree* . The user-supplied buffer "buf" is filled with up to "len" characters of information from the first title record in the database.

All objects (groups and regions) which are to be included in the description to be raytraced must be preprocessed with *rt_gettree* , which returns -1 for failure and 0 for success. This function can be called as many times as required. Be certain to pass the struct rt_i pointer from rt_dirbuild() each time.

To fire a ray at the model, an application structure must be prepared and its address passed to *rt_shootray* . Note that it is mandatory that you provide values for a_ray.r_pt (the starting point of the ray to be fired), a_ray.r_dir (a unit-length direction vector), a_hit (address of user-supplied hit routine), a_miss (address of user-supplised miss routine), a_rt_i (struct rt_i pointer, from rt_dirbuild()), a_onehit (flag controlling stop after first hit), and a_level (recursion level, just for diagnostic printing).

To obtain a report of cpu usage for a portion or portions of your program, frame the statements with calls to *rt_prep_timer* and *rt_read_timer*. Each call to *rt_prep_timer* resets the timing process, after which *rt_read_timer* can be called to get a double which is the elapsed CPU time in seconds since *rt_prep_timer* was last called. In addition, up to "len" bytes of system-specific detailing of resource consumption is placed in the user-supplied buffer "buf".

rt_bomb can be used to exit your program with *msg* printed on the standard error output.

EXIT CODE

All truely fatal errors detected by the library use *rtbomb* to exit with a status of 12.

DEFINITION

RPP – Rectangular parallel-piped. A region of space defined by minimum and maximum values in X, Y, and Z. RPPs are used by librt as the bounding volume for solids.

DISCUSSION

You should study the structures in *raytrace.h* , in particular, the *application* , the *partition* structure and its component structures to get an idea of what information is handed to/from *rt_shootray* . *rt_shootray* may be called recursively as from your *a_hit* routine (good for doing bounced rays). If you only care about the first object hit along the path of the ray, set the *a_onehit* flag in the application structure before calling *rt_shootray* . *rt_shootray* returns the return value of the user-supplied hit or miss function called.

If the ray intersects the model, the a_hit() routine is called with a pointer to the application structure and a pointer to a linked list of ray **partitions** (struct partition). Within each partition are **segment** and **hit** structure pointers for the places that the ray enters and leaves this partition of space. pt_inhit.hit_dist is the parametric distance at which the ray enters the partition, and pt_outhit.hit_dist is the parameteric distance the ray leaves. Note that while the **hit** structure contains hit_point and hit_norm elements, they are not computed by rt_shootray(). If these are needed, they can be filled in by using the RT_HIT_NORM() macro.

Helpful in generation a grid of ray origins, the bounding RPP of the model is computed as *rt_gettree* is called and stored in *rtip->mdl_min* and *rtip->mdl_max* .

EXAMPLE(S)

A program can be loaded as follows:

$ cc –I?? main.c /usr/brl/lib/librt.a –lm

Here is a portion of a hypethetical program which uses the library:

```
#include "vmath.h"
#include "raytrace.h"
main( argc, argv )                              /* m a i n ( ) */
int argc;
char *argv[];
{
        extern int optind;                      /* Used by getopt(3C) */
        extern int do_if_hit(), do_if_miss();   /* Application routines */
        register int h, v;
        int grid_sz;
        struct application ap;          /* Set up for rt_shootray() */
        struct rt_i *rtip;
        ...

        /* Build the directory.  */
        rtip = rt_dirbuild( argv[optind++] );

        /* Load the desired portion of the model. */
        objects = &argv[optind];
        while( argv[optind] != NULL ) {
                rt_gettree( rtip, argv[optind++] );
```

```
        }
        ap.a_hit = do_if_hit;           /* Supply routine for hit */
        ap.a_miss = do_if_missed;       /* Supply routine for miss */
        ap.a_rt_i = rtip;
        ap.a_level = 0;
        ap.a_onehit = 1;                /* Return only closest object hit */

        for(    v = 0;                  /* First scanline.      */
                v < grid_sz;            /* Check for end.       */
                ++v                     /* Next scanline.       */
           ) {
                for( h = 0; h < grid_sz; ++h ) {
                        /* Set up ray origin. */
                        VMOVE( ap.a_ray.r_pt, get_grid( h, v ) );
                        /* Compute ray direction. */
                        VMOVE( ap.a_ray.r_dir, get_dir( h, v ) );
                        VUNITIZE( ap.a_ray.r_dir ); /* Must be unit vector. */
                        (void) rt_shootray( &ap );
                }
        }
}

static int
do_if_hit( ap, PartHeadp )
register struct application *ap;
struct partition *PartHeadp;
{
        register struct partition  *pp = PartHeadp->pt_forw;

        RT_HIT_NORM( pp->pt_inhit, pp->pt_inseg->seg_stp, &(ap->a_ray) );
        RT_HIT_NORM( pp->pt_outhit, pp->pt_outseg->seg_stp, &(ap->a_ray) );
        /* Check for flipped normal and fix (if you intend to use it) */
        if( pp->pt_inflip ) {
                VREVERSE( pp->pt_inhit->hit_normal, pp->pt_inhit->hit_normal );
                pp->pt_inflip = 0;
        }
        if( pp->pt_outflip ) {
                VREVERSE( pp->pt_outhit->hit_normal, pp->pt_outhit->hit_normal );
                pp->pt_outflip = 0;
        }

        /* Do something based on information in partition structure
         *      such as output a pixel to the frame buffer
        */
        light_model( pp->pt_inhit );
        ...
        return  1; /* Report hit to main routine */
}

/*ARGSUSED*/
static int
do_if_miss( ap, PartHeadp )
```

```
register struct application *ap;
struct partition *PartHeadp;
{
        return  0; /* Report miss to main routine */
}
```

SEE ALSO

mged(1B), rt(1B)

DIAGNOSTICS

"rt_malloc: malloc failure", if librt is unable to allocate memory will malloc(). "rt_XXX: read error", if an error or EOF occures while reading from the model database. "unexpected SIGFPE!0 when a floating point error occurs. (The rootfinder traps SIGFPE, but SIGFPE elsewhere is unexpected). "rt_shootray: zero length dir vector0 when the a_ray.r_dir vector is not unit length. "rt_gettree called again after rt_prep!" when an attempt is made to add more sub-trees to the active model after calling rt_prep (or after firing the first ray). "rt_prep: re-invocation" when rt_prep called more than once. "rt_prep: no solids to prep" when there are no valid solids in the model.

AUTHOR(S)

Michael John Muuss

BUGS

The rootfinder detects "hard" cases by taking a SIGFPE and retyring with a slower but more stable algorithm. This is unfortunate.

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

COPYRIGHT

This software is Copyright (C) 1985 by the United States Army. All rights reserved.

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <GED@BRL.ARPA>.

NAME

bw – Black and White image file format and utilities

FORMAT

A *bw* file contains sequences of pixels. Each pixel is stored as one **unsigned char**, and thus ranges in intensity from 0 (black) to 255 (white). The first pixel in a *bw* file is the lower left corner of the image. The pixels proceed from left-to-right across each scanline, with scanlines being written from the bottom to the top of the image.

By convention, *bw* images are usually square, and thus their dimensions can be determined by the file size. If this is not the case, often only the file width need be known by a program reading the file. History has left us with two "standard" sizes, 512x512 and the "hires" 1024x1024. A common practice for other file sizes is to include the file width in the filename.

At some time in the future *bw* files will probably get self-typing headers so that parameters such as their size can be automatically determined by programs.

UTILITIES

bw-fb display a .bw format image on a framebuffer.
fb-bw produce a .bw file from a framebuffer image.
pix-bw convert a color .pix file to a .bw file.
bwstat gives statistics concerning a .bw file.
bwhist displays a histogram on a framebuffer.
bwmod a filter to shift and scale pixel intensities.
bwdiff yields the difference between two .bw files.
bwrect extracts an arbitrary rectangle from a .bw file.
bwcrop maps any quadrilateral in a .bw file into any rectangle.
bwrot rotates, reverses, or inverts a .bw file.
bwscale scales a .bw file larger or smaller.
bwfilter a variety of 3x3 kernel filters for .bw images.
bw3-pix merges three .bw files into one .pix file.
pix-bw3 separates .pix file RGB colors into three .bw files.

FILES

file.bw

SEE ALSO

brlcad(1), pix(5), and above named programs.

NAME

pix – Color RGB image file format and utilities

FORMAT

A ***pix*** file contains sequences of pixels. Each pixel is stored as three **unsigned chars**, first red, then green, then blue. The intensity of each color ranges from 0 ("off") to 255 ("on"). The first pixel in a ***pix*** file is the lower left corner of the image. The pixels proceed from left-to-right across each scanline, with scanlines being written from the bottom to the top of the image.

By convention, ***pix*** images are usually square, and thus their dimensions can be determined by the file size. If this is not the case, often only the file width need be known by a program reading the file. History has left us with two "standard" sizes, 512x512 and the "hires" 1024x1024. A common practice for other file sizes is to include the file width in the filename.

At some time in the future ***pix*** files will probably get self-typing headers so that parameters such as their size can be automatically determined by programs.

UTILITIES

pix-fb display a .pix format image on a framebuffer.
fb-pix produce a .pix file from a framebuffer image.
pix-bw convert .pix file to a .bw file.
ap-pix convert an Applicon Color Plotter file to a .pix file.
pixtile combine several small .pix files into one large one.
pixstat gives statistics concerning a .pix file.
pixhist displays a color histogram on a framebuffer.
pixhist3d display three views of an RGB color space histogram on a framebuffer.
pixhist3d-pl unix plot of RGB color space utilization.
pixdiff shows which pixels/colors differ between two .pix files.
pixrect extracts an arbitrary rectangle from a .pix file.
pixrot rotate, reverse, or invert a .pix file.
pixscale scale a .pix file up or down.
pixfilter a collection of 3x3 kernel filters for .pix files.
bw3-pix merges three .bw files into one .pix file.
pix-bw3 separates .pix file RGB colors into three .bw files.

FILES

file.pix

SEE ALSO

brlcad(1), bw(5), and above named programs.

NAME

plot3 – 3-D and color graphic plotting interface

DESCRIPTION

Files of this format are produced by programs such as *pixhist3d-pl*(1), using the routines described in *libplot3* (3) and are interpreted for various devices by standard programs described in *plot* (1G), as well as additional programs such as *plot-fb*(1). The format of output from *libplot3* differs from that of *libplot* only in the addition of 3-D primatives and color. A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the **x** and **y** values; each value is a signed 16-bit integer. These integers are written with the most significant byte first. Three-dimensional data have an additional **z** value, which is assumed to be zero for two-dimensional instructions. The last designated point in an **l**, **m**, **n**, **p**, **L**, **M**, **N**, or **P** instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *libplot3* (3).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes. See *plot*(1G).

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius. Circles and arcs are not supported on all devices.

e erase: Start another frame of output.

f linmod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are "dotted", "solid", "longdashed", "shortdashed", and "dotdashed". Effective only for the **–T4014** and **–Tver** options of *plot* (1G) (TEKTRONIX 4014 terminal and Versatec plotter).

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

The following were added at BRL. Three-dimensional primitives have their **z** values ignored when displayed on two-dimensional devices.

C color: The next three bytes give respectively the intensities (0..255) of the three color components (additive colors red, green, and blue if a terminal; subtractive colors cyan, magenta, and yellow if hardcopy) to be used for subsequent plotting. The initial color is assumed to be (255,255,255), which is white on a terminal and black on hardcopy. Color specifications are ignored for black-and-white devices.

M 3move: The next six bytes give a new current point.

N 3cont: Draw a line from the current point to the point given by the next six bytes. See *plot* (1G).

P 3point: Plot the point given by the next six bytes.

L 3line: Draw a line from the point given by the next six bytes to the point given by the following six bytes.

S 3space: The next six bytes give the minimum corner of the plotting area; the following six give the maximum corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *plot* (1G). The upper limit is just outside the plotting area, except for the Megatek which displays the entire range of possible coordinates. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	space(0, 0, 3120, 3120);
Versatec plotter	space(0, 0, 2048, 2048);
Megatek display	space(-32768, -32768, 32767, 32767);

SEE ALSO

libplot3(3), plot(3), plot(5).
graph(1G), plot(1G) in the *UNIX System User Reference Manual.*

AUTHOR

Douglas A. Gwyn made the 3-D and color extensions to UNIX-plot.

SOURCE

SECAD/VLD Computing Consortium, Bldg 394
The U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005

BUG REPORTS

Reports of bugs or problems should be submitted via electronic mail to <CAD@BRL.ARPA>.

The following is a description of the libpkg routines kindly provided by Peter B. Shames, updated to reflect BRL extensions since the first writing. It is intended that an RFC be published at some point, but for now this (plus the source code) is all the documentation we have. In the CAD distribution, the remote framebuffer interface (libfb/if_remote.c) and server (rfbd), communicate using this protocol.

Contributors:

BRL U.S.Army Ballistic Research Laboratory
STScI Space Telescope Science Institute

==========

PKG Interface Description 26 Sept 1986
==========
PMB Shames

[Updated: 14 Nov 1986
PC Dykstra]

Overview

A prototype interface that implements a fairly simple message passing abstraction between cooperating processes running over TCP connections has been developed by Mike Muuss, Charles Kennedy, and Phillip Dykstra at BRL. Some experiments have been run at STScI to implement this interface on a DECnet link with good success. A few changes have been suggested to improve network throughput for data flows that consist of short message traffic or byte-stream traffic.

This note describes the motivation for developing such an interface and describes the subroutine interfaces and functionality. The interface routine descriptions and portions of the textual descriptions have been taken directly from the source code provided by Muuss and Kennedy which has been the only available description to date.

Why a message interface?

The interface - PKG provides a simple interface for managing client/server connections in a distributed environment. A server task is initialized to wait for incoming requests from clients. Clients open a connection to the server task which executes on some host processor waiting for incoming requests. Messages with identifying type codes are passed between processes, with optional appended data blocks. The interface is non-blocking and accepts arbitrary data messages of arbitrary length.

An example - interaction with a remote data archive may require request for services, user authentication, permission granting, index hierarchy queries - with manipulation of results, archive request processing, delivery of output products - electronically or otherwise. The archive is a server that responds to service requests from clients, how many may be served at one time and whether public access and private services are supported is an implementation issue. Since no direct interactive

support is required (if smart client agents are in use) the message paradigm can support the requirements.

A [counter] example - Athena's XWindow uses a client-server model to provide screen/window management services for distributed processes. XWindow is very permissive of different window management policies and provides hooks for implementing various user interfaces. At the lowest level, XWindow requires a reliable duplex byte-stream, which can be implemented in many operating environments. Because XWindow wants to manage all process-window communications directly, it builds its own block stream protocol to manage the connection. This permits the interactive update required by some processes, but basically is the same paradigm as PKG, defined within the window system.

PKG Interface description

The PKG interface implements an client-server network connection that multiplexes synchronous and asynchronous messages across stream connections. Connection control is provided by open/ close routines, where the client authentication is handled prior to establishing the connection. Server side actions include initialization, client connection, and client request servicing. Client side actions include server request, reply processing, possible connection negotiation, and data transfer processing.

A connection is established when a client issues an open to a server, running on some host. Which hosts provide required services must be determined prior to connection requests, using some service directory. Default connection processing is stream oriented, where a process may get data as it is available on the connection. A blocking read may also be issued, waiting for any complete message, or just for messages of a specified type. The connection remains open until explicitly closed by either client or server.

Once the connection is established it may be used to process a full-duplex asynchronous byte stream, or a buffered, synchronous query/reply exchange. Message type multiplexing supports user actions based upon message type, where the multiplexing is handled in the interface. Different message type streams may operate using different flow control over the same connection. Multiplexing is handled in the interface by pre-pending a header block to the message before transmitting it to the reader. Data is transmitted as received, but the header is transmitted in Internet network byte/bit order.

An package connection may incur significant overhead if many short messages are passed between processes, so a buffered stream connection may be requested. The application calls the pkg_stream interface rather than the usual pkg_send request, and PKG builds a message buffer and passes it to the client as it gets filled. A stream buffer may be flushed with an explicit call to the pkg_flush routine.

PKG Interface Definition

Control Interfaces -

PKG_OPEN - Open a connection to a host

pkg_open(host, service, protocol, uname, passwd, pkg_switch, errlog)
returns ptr_to_connection

We are a client, make a connection to a server running
on some host. The return value is the connection handle.
Protocol, username, and password are optional.

PKG_CLOSE - Close a connection

pkg_close(ptr_to_connection) returns VOID

We are a client or server wishing to gracefully close
a connection.

PKG_TRANSERVER - Become a one-time network server on the already accepted connection.

pkg_transerver(pkg_switch, errlog) returns ptr_to_connection

Given an established connection, this routine creates a
pkg_conn structure for it and initializes it to use the
given pkg_switch and error logging function. This routine
is needed by servers which get created by some other
process after the connection is established. Processes
started by the UNIX "inetd" are one example.

PKG_PERMSERVER - Create a network server and listen for connections

pkg_permserver(service, protocol, backlog) returns listen_file_desc

The service port is determined and a listen is hung on
the port that is bound to the socket. The return value
is the file descriptor of the socket.
For this kind of server, pkg_getclient is used to
accept connections from clients.

PKG_GETCLIENT - Used by a server to accept a client connection

pkg_getclient(listen_file_desc, pkg_switch, errlog, nodelay_flag)
returns ptr_to_connection or NULL or ERROR

If possible, the connection request from the client is
accepted, a pkg_connection block is created and a ptr to
it is returned to the server. The server may request
non-blocked service by setting the nodelay_flag TRUE.

I/O Interfaces -

PKG_GET - Read bytes from connection, assembling message

pkg_get(ptr_to_connection) returns message_arrived, more_data_coming or ERROR.

Get waits until a message header is received and the calls the user specified message handler for that message type. This routine should be called by a program whenever there is data waiting on a connection and the program is not otherwise waiting on a specific message type. The routine will return directly to the caller if no header is available or if only a partial message is received, otherwise it calls the user specified message type handler.

PKG_SEND - Send a message on the connection

pkg_send(message_type, data_buff, buff_len, ptr_to_connection) returns bytes_sent or ERROR

Send constructs a message header for the data buffer and transmits it on the connection. If only part of the message can be sent the actual number of bytes transmitted returned.
Any data in the stream buffer is first flushed. If the stream buffer needs flushing, and the message is less than MAXQLEN (currently 512) bytes long, and sufficient room is left in the stream buffer, this message gets "piggybacked" on by copying it to the buffer before flushing.
If available, "writev" is used to send the header and data with one syscall.

PKG_2SEND - Send a two part message on the connection

pkg_2send(message_type, data_buff1, buff1_len, data_buff2, buff2_len, ptr_to_connection) returns bytes_sent or ERROR

2Send performs the same job as pkg_send except the data comes in two parts. This is often the case for say a command followed by user data. User data copies can be avoided while still allowing a single trip in and out of the kernel on the receiving end.

PKG_STREAM - Send a buffered data stream on the connection

pkg_stream(message_type, data_buff, buff_len, ptr_to_connection) returns bytes_sent or ERROR

Pkg_stream connections provide a low network overhead connection where interactive responsiveness is not required.
Connections have a PKG_STREAMLEN (currently 4096) byte stream buffer. If the current message is less than MAXQLEN (currently 512) bytes in length, and space

remains for it, it will be appended onto the end of this buffer. Otherwise the buffer is implicity flushed by sending this message via PKG_SEND.

PKG_FLUSH - Flush the stream buffer out on the connection

pkg_flush(ptr_to_connection) returns bytes_sent or ERROR

Flush takes the current contents of the stream buffer being constructed by stream and flushes it to the connection. This allows programs explicit control over the stream interface where required.

PKG_BLOCK - Wait for a complete message of any type

pkg_block(ptr_to_connection) returns message_arrived or ERROR

This routine blocks, waiting for a complete message of ant type to arrive on the connection. The user message handler is called to process the message. This routine can be called in a loop waiting for asynchronous messages or for the arrival of messages of uncertain type.

PKG_WAITFOR - Wait for a message of specific type, return in user buffer

pkg_waitfor(message_type, user_buff, buff_len, ptr_to_connection) returns buff_len or ERROR

This routine does a blocking read on the connection until a message of message_type is received. Asynchronous messages and messages of other types are processed while this routine waits. The message is returned in the user's buffer.

PKG_BWAITFOR - Wait for a message of specific type, return in allocated buffer

pkg_bwaitfor(message_type, ptr_to_connection) returns ptr_to_buffer or ERROR

This routine does a blocking read on the connection until a message of message_type is received. Asynchronous messages and messages of other types are processed while this routine waits. The message is returned in a newly allocated buffer that the caller must free.

User Message Handler Interface -

PKG_SWITCH - Table of User Message Handler Pointers

```
struct pkg_switch {
    unsigned short  pks_type; /* Type code */
```

```
        int   (*pks_handler)();     /* Message Handler */
        char *pks_title;        /* Description */
};
```

pks_handler(ptr_to_connection, message_buff) returns ignored int.

The user may specify handlers for one or more message types. When these routines are called they are passed a pointer to the connection and a pointer to a buffer that contains the message. The user message handler routine is resonsible for freeing the message buffer after processing it. If there is no message handler for that message type the message is discarded and the buffered freed. The switch routine uses an external message handler array, created by the caller, that defines message types, handler entry points, and a message descriptor field.
The pkg_switch list which is passed to some of the above functions is a NULL terminated array of structures containing the message type, a pointer to an function which handles messages of that type, and a descriptive string. A seperate pkg_switch array is kept for each connection so that several entirely different PKG conversations can be carried out by a single application at the same time.

PKG_CONN - Connection block structure

```
struct pkg_conn {
        pkg_file_desc, pkg_magic, mess_len, ptr_to_buffer,
        curr_buffer_pos, pkg_switch, errlog, stream_buff,
        curr_stream_pos
};
```

Connection block structures are created by pkg_open or pkg_getclient when a connection is established, and are freed when a connection is closed. This connection block is used during all I/O requests and contains the pointer to the start of the current buffer and to the current position in the buffer for incomplete read or writes. The mess_len field specifies the number of bytes expected by get.
The pkg_switch array, and a pointer to an error logging function for this connection is included. The stream buffer is also contained in the structure.

See pkg.h for the actual names of the structure elements.

Implementation Notes

At least two implementations of the PKG interface have been done, one based on TCP/IP connections, the other based on DECnet connections. Future releases of PKG will be layered on top of a virtual stream library known as NET.

The implementation running on 4.2BSD UNIX takes advantage of

the "writev" syscall which can output the header and data in a single syscall/TCP_PUSH cycle. The advantage of the short data copy is realized only when writev can not be used, or when a pkg_flush is not required (a la PKG_STREAM).]

Future Developments

Further development of the lower levels of the interface will be provided so that this message passing abstraction can be provided on top of both DECnet and TCP/IP network layers, with other implementations provided as required. The program interface layer is to be as described in the body of this note, the lower layers will interface to the network layers using whatever subroutine interfaces are most appropriate for the system hosting the interface. A simple read/write/open/close abstraction may be provided at a lower layer to further isolate the network layers from the message handling layer.

Where it is necessary to cross network protocol boundaries, message relay processes will be created to reside on a Janus host that straddles the two domains. Domain routing and relay point information will be maintained above the network layer within the extended PKG interface. Clients and servers will need to determine server host id information from some available database. Server names must either carry domain information or such information must be available as part of the server database. Within the extended PKG interface routing information for domain boundary transits must be maintained, and appropriate Janus host systems connected to provide relay services.

It is not anticipated that full asynchronous I/O implementation will work particularly well across domain boundaries, but all other forms of message transfer should work quite well. As already noted, buffered stream I/O may be prefectly satisfactory for many applications where full interaction is not required. Where applicable, this mode of interface is to be prefered over full asynchronous I/O. Further development of this interface layer on other protocol families is possible, perhaps even on low level networks like BITNET and certainly on connection oriented X.25 or OSI networks.

3.

Understanding the Preparation and Analysis of Solid Models

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.

December 2, 1986

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,[1] 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

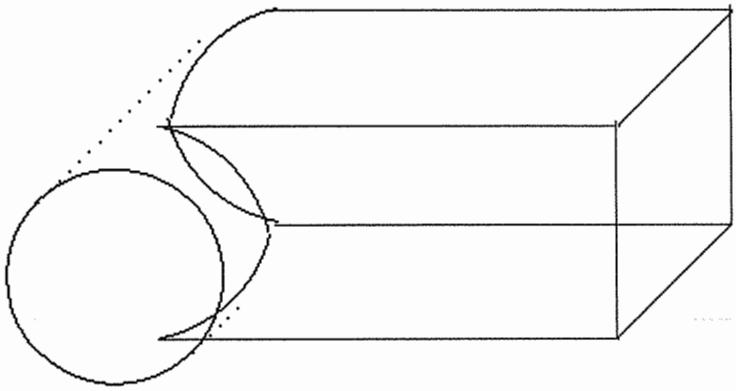


Figure 1.1 — Wireframe Ambiguity

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.

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.[1] 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.[2] 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.

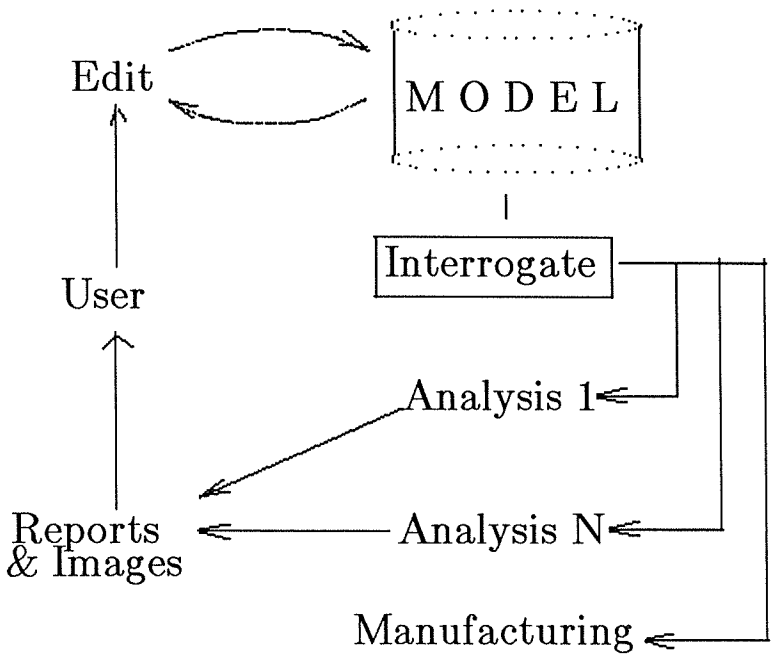


Figure 1.2 — The Design Loop

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.[3]

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.[4] 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[5] 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,[1] 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[6,7] from Draper Laboratories and TIPS-1[8-12] 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.[13,14] 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),[15] PADL-1,[16,17] PADL-2,[18] Series 7000 SMS (Auto-Trol), GMSolid,[19,20] SynthaVision (MAGI)[21,22] (SynthaVision is also marketed as ICEM by CDC, and Solids Modeling II by Applicon), MGED (BRL),[23,24] 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,[25] Bezier patches,[26] splines,[27] etc. Some examples of B-rep systems (not all of which qualify as solid modelers) are: Alpha_1 (U. Utah),[28] Build-2 (U. Cambridge),[29,30] CADD (McAuto), CATIA (Dassault/IBM), Compac (Technical U. Berlin),[31-35] Euclid (Matra Datavision),[36] Euklid (Fides Co.),[37,38] Glide (CMU),[39,40] Medusa (Cambridge Interactive Systems, Ltd.), PATRAN (PDA Engineering), Romulus (Shape Data Ltd, Evans&Sutherland),[41] 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 $\cap$ B-spline results in polygons rather than another B-spline. In a boundary representation closed under the set of boolean operations, B-rep $\cap$ 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.[42]

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.

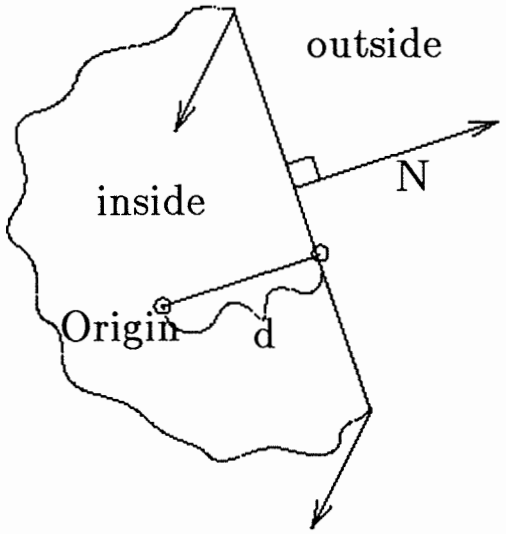


Figure 1.3 — Definition of a Halfspace

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 $\mathbf{N}$ normal to the surface of the plane with unit length (i.e. $\mathbf{N}=1$) 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 $\mathbf{X}$ is contained within the halfspace if $\mathbf{N} \cdot \mathbf{X} \leq d$. Points lying on the surface of the plane are considered to be inside the halfspace.

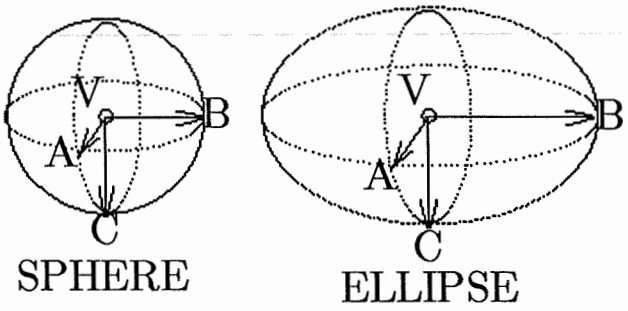


Figure 1.4 — Ellipsoid

1.6.2. Ellipsoid (ELL)

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

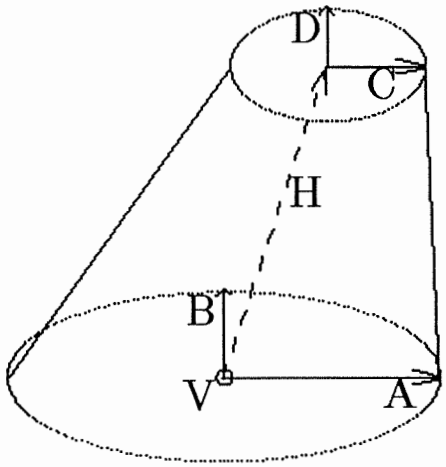


Figure 1.5 — Truncated General Cone

1.6.3. Truncated General Cone (TGC)

Referring to figure 1.5, a truncated general cone can be defined by specifying a vector $\mathbf{V}$ from the origin to the center of the "lower" ellipse, two perpendicular vectors $\mathbf{A}$ and $\mathbf{B}$ which define the orientation and eccentricity of the lower ellipse, a vector $\mathbf{H}$ which defines the height and slant of the cone, and two more perpendicular vectors $\mathbf{C}$ and $\mathbf{D}$ which define the orientation and eccentricity of the upper ellipse.

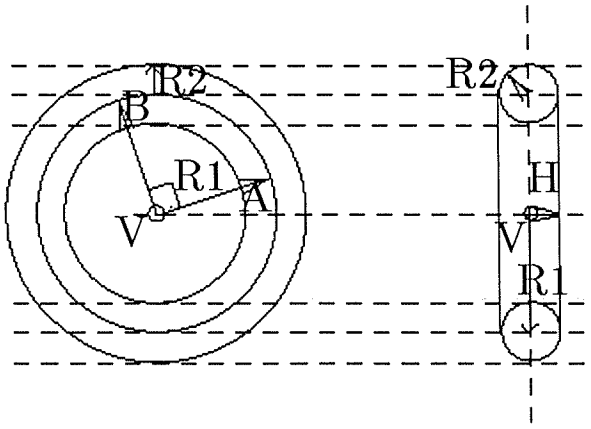


Figure 1.6 — Torus

1.6.4. Torus (TOR)

Referring to figure 1.6, a torus can be defined by specifying a vector **V** from the origin to the center of the torus, three mutually perpendicular vectors **A** and **B** and **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.

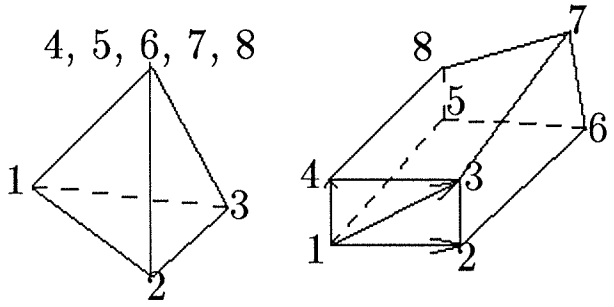


Figure 1.7 — Convex Polyhedron (ARB8)

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 **V** from the origin to the first vertex and seven additional vectors $\mathbf{R}_{i\ i=1..7}$ from **V** to the other vertices. Vectors $\mathbf{R}_i$ may coincide or have length=0, $\mathbf{R}_{i\ i=1..7} \geq 0$. Examples of this type of solid are the wedge, box, pyramid, etc.

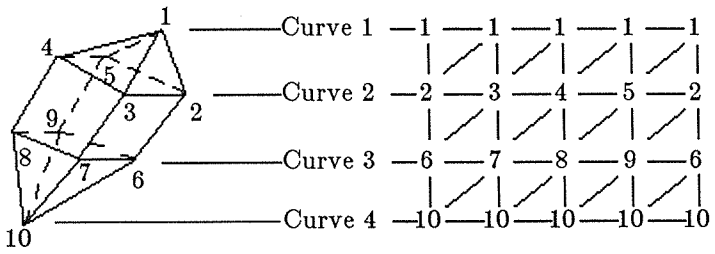


Figure 1.8 — Faceted Polyhedron (ARS)

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,[27,43] while a basis for good computer implementations can be found in Riesenfeld[44] and Forrest,[45] so they will not be described further. Details of implementations to design sculptured surfaces using the B-spline representation can be found in Cobb[46] and Stay.[47]

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.[48]

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 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.

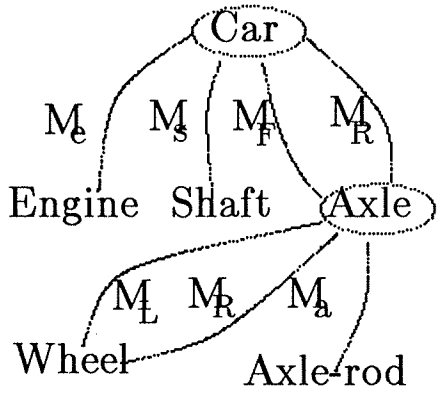


Figure 1.9 — Directed Acyclic Graph for Simple Car

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

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

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

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

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

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

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

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

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

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

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,[49] and also see pp. 46-88 in Rogers and Adams.[50]

1.9. References

For reviews of the solid modeling field in general, see Requicha,[1] Johnson,[51] and Baer.[52] For more information on current solid modelers, the interested reader is referred to Wyvil,[53] Boyse,[19] Muuss,[24] and Brown.[18]

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,[5] which was adopted and documented by a tri-service Joint Technical Coordinating Group for Munitions Effectiveness[54,55] and also resulted in a number of other papers being published.[21] Further development at BRL evolved the MAGIC ray-tracing code into the GIFT (Geometric Information from Targets) ray-tracing code,[56,57] while MAGI pursued in-house developments which eventually resulted in the commercial Synthavision system.[22,58]

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,[23] 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.[24] 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[59] 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[24] 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[60] 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,[61] with all the software written in the "C" programming language.[60] All communications software is based upon the TCP/IP protocol suite developed by DARPA[62] 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[63] 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.

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

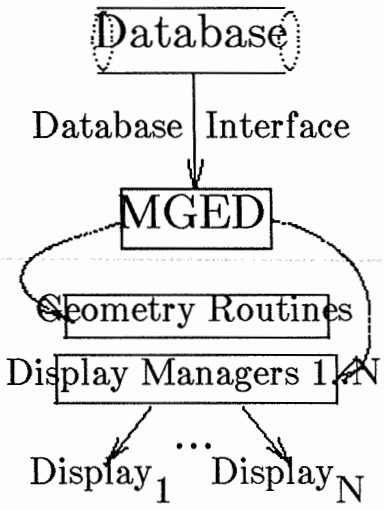


Figure 2.1 — Object-Oriented Design

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.

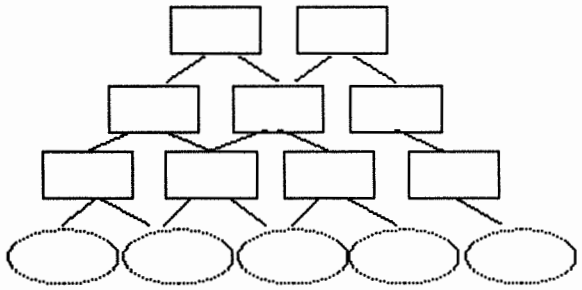


Figure 2.2 — Hierarchy

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.

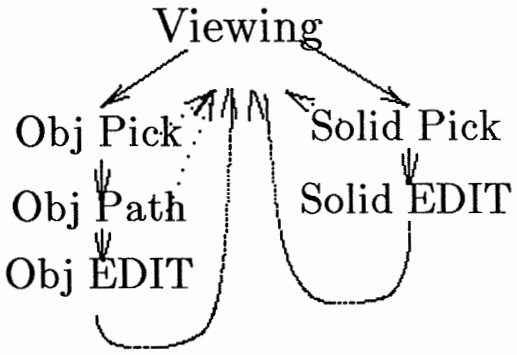


Figure 2.3 — Editor State Transitions

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 × 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 edge 12
move edge 23
move edge 13
move edge 14
move edge 24
move edge 34
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.[64,65] 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 hyper-patches.

While very recent research has begun to explore techniques for intersecting cylinders, cones, and planes with geometry,[66] 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[5] in 1967. Extensions to the early developments were undertaken by a DoD Joint Technical Coordinating Group effort, resulting in publications in 1970[54] and 1971.[55] 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[67] thesis in 1979 being a notable early work. One of the central papers in the ray-tracing literature is the work of Whitted.[68] Model sampling techniques can be improved to provide substantially more realistic images by using the "Distributed Ray Tracing" strategy.[69] For an excellent, concise discussion of ray-tracing, consult pages 363-381 of Rogers.[70]

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.[71]

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

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

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 $\mathbf{D}$ is of unit length, i.e. $|\mathbf{D}| = 1$.

3.4. Ray/Halfspace Intersection

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

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 $\mathbf{N}$:

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

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

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

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

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

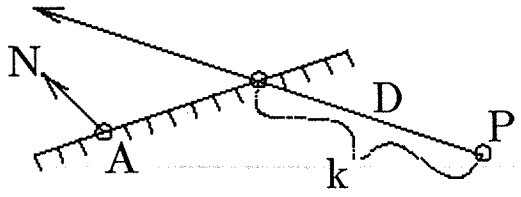


Figure 3.1 — Ray/Halfspace Intersection

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 $\mathbf{X}$ on the ray may be expressed as $\mathbf{X}=\mathbf{P}+k\mathbf{D}$. 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 $\mathbf{D}\cdot\mathbf{N}<0$, is *parallel* to the plane when $\mathbf{D}\cdot\mathbf{N}=0$, and *exits* otherwise. Find the *entry* point farthest away from the starting point $\mathbf{P}$, 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 $\mathbf{P}$, i.e. it has the smallest value of k among the exit points. The ray exits the solid here.

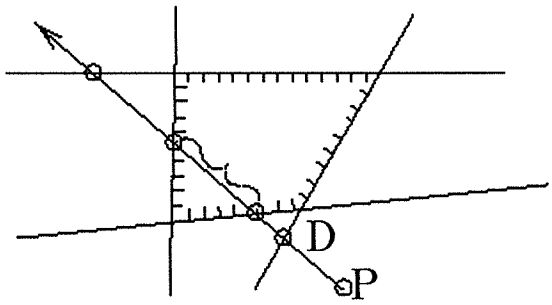


Figure 3.2 — Ray/ARB Intersection

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 $\mathbf{V},\mathbf{A},\mathbf{B},\mathbf{C}$, where $\mathbf{V}$ is the center of the ellipsoid, and $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ 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 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 $X=S(R(X-V))$, where

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

and

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

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 $P+\hat{k}*D$. The mapping between regular and "hat" space for vectors is:

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

and for points is:

$$P=S(R(P-V))$$

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

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

and

$$W=S(R(W-V))$$

In order to find the point 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 W, giving

$$(P_x+\hat{k}*D_x)^2 + (P_y+\hat{k}*D_y)^2 + (P_z+\hat{k}*D_z)^2 = 1$$

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

$$(D_x^2+D_y^2+D_z^2)\hat{k}^2 + 2*(D_xP_x+D_yP_y+D_zP_z)\hat{k} + P_x^2+P_y^2+P_z^2-1 = 0$$

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

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

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

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

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

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

$$W=S(R(W-V))$$

can be rearranged in terms of W

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

Then, substituting in the parametric form of W

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

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

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

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

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

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 D and 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 N:

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

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

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

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

$$\begin{aligned} N&=[S^{-1}*R]^{-1}(\hat{N}) \\ &=R^{-1}*S(\hat{N}) \\ &=R^{-1}(S(S(R(W-V)))) \end{aligned}$$

Note that $N \neq 1$.

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.

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

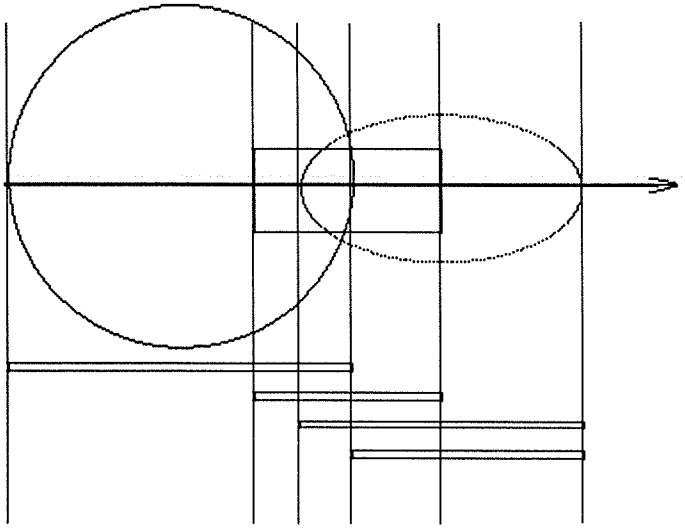


Figure 3.3 — Boolean Evaluation

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 $\mathbf{V}$ to line $\mathbf{P} + k^*\mathbf{D} \leq r$, i.e.

$$(\mathbf{V}-\mathbf{P})\times\frac{\mathbf{D}}{\mathbf{D}} \leq r$$

When $\mathbf{D} = 1$, 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.

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[72] 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 ) {
```

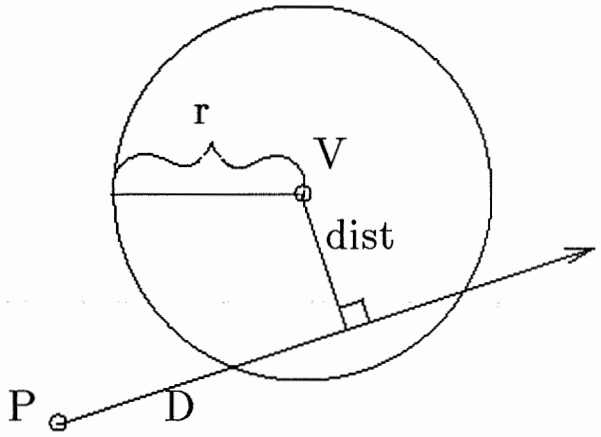


Figure 3.4 — Bounding Sphere

```
        /* 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.[73-75]

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 (**P**) which is the starting point of the ray to be fired, the vector a_ray.r_dir (**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 **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,[69] 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.[2]

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.[76] 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.[77] 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

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

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

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

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

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

These computational techniques can be extended[78] 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.[79] 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,[1] recent research is beginning to provide strategies for implementing them,[80] 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.[81] 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.[82] 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.[83,84] 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,[85] 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.[2] The Vehicle Code System (VCS)[86,87] 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.[63] 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.[2]

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.[88] 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,[89] 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.[90] BRL has developed a predictive millimeter wave signature model[91] 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.[92] While standard radars only report target backscatter and range information, SAR techniques can resolve distinct scattering regions of a target.[90] 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.[90] 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

1. A. A. G. Requicha, H. B. Voelcker, "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications* **2**(2) pp. 9-24 (March 1982).
2. 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).
3. P. H. Deitz, *The Future of Army Item-Level Modeling,* Army Operations Research Symposium XXIV, Ft. Lee VA (8-10 October 1985).
4. P. H. Deitz, *Solid Modeling at the US Army Ballistic Research Laboratory,* Proceedings of the 3rd NCGA Conference (13-16 June 1982).
5. MAGI Inc, *A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles,* MAGI Report 6701, AD847576 (August 1967).
6. J. H. Laning, D. A. Lynde, V. Moreggia, *SHAPES User's Manual,* Charles Stark Draper Laboratory, Inc., Cambridge MA (May 1973).
7. J. H. Laning, 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 (April 9-11, 1979).
8. N. Okino, Y. Kakazu, 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, (1973).
9. N. Okino et al., *TIPS-1, '77 Version,* Institute of Precision Engineering, Hokkaido University, Sapporo Japan (March 1978).
10. N. H. Okino, H. Kubo, Y. Kakazu, "TIPS-1: practical approaches for an integrated CAD/CAM System," *Proceedings SME Conference on CAD/CAM,* (1973).
11. N. H. Okino, et al., "Graphics processor for computer-aided design," *Proceedings of International Conference on Precision Engineering,* pp. 366-371 (1974).
12. N. H. Okino, H. Kubo, Y. Kakazu, "An integrated CAD/CAM system: TIPS-2," *Proceedings Prolomat Conference,* (1976).
13. G. Kedem, *Computer Structures and VLSI Design for Curve-Solid Classification,* Siggraph '85 Tutorial "VLSI for Computer Graphics", San Francisco CA (July 23, 1985).
14. 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).
15. W. Fitzgerald, F. Gracer, R. Wolfe, "GRIN: Interactive Graphics for Modeling Solids," *IBM Journal of Research and Development* **25**(4) pp. 281-294 (July 1981).
16. 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).
17. 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).
18. Brown, "PADL-2: A Technical Summary," *IEEE Computer Graphics and Applications* **2**(2) pp. 69-84 (March 1982).
19. J. W. Boyse, J. E. Gilchrist, "GMSolid: Interactive Modeling for Design and Analysis of Solids," *IEEE Computer Graphics and Applications* **2**(2) pp. 27-40 (March 1982).
20. S. D. Roth, *Ray Casting as a Method for Solid Modeling,* Research Publication GMR-3466,

General Motors Research Laboratories, Warren MI (October 1980).

21. R. Goldstein, L. Malin, "3-D Visual Simulation," *Simulation* **16**(1) pp. 25-31 (1971).
22. R. Goldstein, L. Malin, "3D Modelling with the Synthavision System," *CAD/CAM Systems (Proceedings of the First Annual Conference Computer Graphics),* pp. 244-247 (9-11 April 1979).
23. E. P. Weaver, M. J. Muuss, *Interactive Construction of COM-GEOM Targets,* BRL Spring Technical Conference (1980).
24. M. J. Muuss, K. A. Applin, R. J. Suckling, G. S. Moss, E. P. Weaver, C. Stanley, *GED: An Interactive Solid Modeling System for Vulnerability Analysis,* BRL Technical Report ARBRL-TR-02480, NTIS AD# A-126-657 (March 1983).
25. S. A. Coons, *Surfaces for Computer-Aided Design of Space Forms,* Tech. report MAC-TR-41, Project MAC, MIT, NTIS AD# 663-504, Cambridge MA (June 1967).
26. P. E. Bezier, *Mathematical and Practical Possibilities of UNISURF,* Academic Press, New York (1974).
27. C. deBoor, *A Practical Guide to Splines,* Springer-Verag, New York (1978).
28. E. Cohen, "Mathematical Tools for a Modelers Workbench," *IEEE Computer Graphics and Applications,* (October 1983).
29. I. C. Braid, "New Generations in Geometric Modeling," *Proceedings of Geometric Modelling Project Meeting,* P-78-GM-01, CAM-I, Inc., (March 14-16, 1978).
30. I. C. Braid, R. C. Hillyard, and I. A. Stroud, "Stepwise Construction of Polyhedra in Geometric Modelling," pp. 123-141 in *Mathematical Methods in Computer Graphics and Design,* ed. K. W. Brodie, Academic Press, London (1980).
31. F. L. Krause and Vassilakopoulos, "A way to Computer Supported Systems for Integrated Design and Production," *IFIPS CAD Working Conference,* (February 1976).
32. 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).
33. G. Spur and F. L. Krause, "Stages of integration for computer supported design and manufacturing process planning," *IFTOMM,* (October 1976).
34. G. Spur and J. Gausemeier, "Processing of Workpiece Information for Producing Engineering Drawings," *Proceedings 16th International Machine Tool Design and Research Conference,* pp. 17-21 (1975).
35. G. Spur, "Status and Further Develoment of the Geometric Modelling System COMPAC," *Proceedings Geometric Modelling Project Meeting,* pp. 1-35 P-78-GM-01, CAM-I, Inc., (March 14-16 1978).
36. Y. J. Bernascon and J. M. Brun, *Automated Aids for the Design of Mechanical Parts,* Tech. Paper MS75-508, Society of Manufacturing Engineers (1975).
37. M. Engeli, "A language for 30 graphics applications," *International Computer Symposium '73,* North-Holland Press, (1974).
38. M. Engeli and V. Hrdliczka, *EUKLID: eine Einfuhrung,* Fides Co., Zurich (1974).
39. 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).
40. C. Eastman and K Weiler, "Geometric Modelling Using the Euler Operators," *Proc. First Annual Conference on Computer Graphics in CAD/CAM Systems,* pp. 248-259 (April 9-11 1979).
41. P. Veenman, "ROMULUS — The Design of a Geometric Modeller," pp. 127-152 in *Geometric Modelling Seminar,* ed. W. A. Carter, P-80-GM-01, CAM-I, Inc., Bournemouth UK (November 1979).

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

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

44. R. F. Riesenfeld, E. Cohen, 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).

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

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

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

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

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

50. D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics,* McGraw-Hill, New York (1976).

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

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

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

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

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

56. L. W. Bain Jr., M. M. Reisinger, *The GIFT Code User Manual; Volume I, Introduction and Input Requirements (U),* BRL Report No. 1802, AD #A078364 (July 1975).

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

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

59. 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).

60. D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "UNIX Time-Sharing System: The C Programming Language," *Bell Sys. Tech. J.* **57**(6) pp. 1991-2019 (1978).

61. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Bell Sys. Tech. J.* **57**(6) pp. 1905-1929 (1978).

62. 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).

63. 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).

64. J. Amanatides, "Ray Tracing with Cones," *Computer Graphics (Proceedings of Siggraph '84)*

18(3)(July 1984).

65. D. B. Kirk, "The Simulation of Natural Features Using Cone Tracin," pp. 129-144 in *Advanced Computer Graphics*, ed. T. L. Kunii, Springer-Verlag (1986).
66. J. T. Kajiya, "New Techniques for Ray Tracing Procedurally Defined Objects," *Transactions of Graphics* **2**(3) pp. 161-181 (July 1983).
67. D. S. Kay, *Transparency, Refraction, and Ray Tracing for Computer Synthesized Images*, Cornell Univ (Jan 1979).
68. J. T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM* **23**(6) pp. 343-349 (June 1980).
69. Cook, Porter, Carpenter, "Distributed Ray Tracing," *Computer Graphics (Proceedings of Siggraph '84)* **18**(3) pp. 137-145 (July 1984).
70. D. F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, New York (1985).
71. M. A. J. Sweeny, R. H. Bartels, "Ray Tracing Free-Form B-spline Surfaces," *IEEE Computer Graphics*, (February 1986).
72. M. L. Cyrus and J. Beck, *Computer Graphics: Two- and Three-Dimensional Clipping*, Air Force Report AFHLR-TR-77-14, NTIS AD# A043017 (May 1977).
73. A. S. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications* **4**(10)(1984).
74. A. Fujimoto, et al., "Accelerated Ray Tracing," *Proceedings of CG Tokyo '85*, (April 1985).
75. A. Fujimoto, C. G. Perrott, and K. Iwata, "Environment for Fast Elaboration of Constructive Solid Geometry," pp. 20-33 in *Advanced Computer Graphics*, ed. T. L. Kunii, Springer-Verlag (1986).
76. 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).
77. G. B. Thomas Jr., *Calculus and Analytic Geometry*, Addison-Wesley (February 1972).
78. G. A. Blass, *Theoretical Physics*, Appleton-Century-Crofts, New York (1962).
79. L. G. Lamit and V. Paige, "The Influence of CADD on Teaching Traditional Descriptive Geometry and Orthographic Projection," pp. 473-484 in *Advanced Computer Graphics*, ed. T. L. Kunii, Springer-Verlag (1986).
80. J. Hashimoto, H. Fukushima, and W. Kowaguchi, "High Performance CAD System with Full Automatic Dimensioning and Multi-Modeling Features Based on Engineering Workstation," pp. 358-378 in *Advanced Computer Graphics*, ed. T. L. Kunii, Springer-Verlag (1986).
81. 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 (February 1981).
82. 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**(18-20 May 1982).
83. 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# B055277 (January 1981).
84. D. A. Ringers and F. T. Brown, *SLAVE (Simple Lethality and Vulnerability Estimator) Analyst's Guide*, BRL Technical Report ARBRL-TR-02333, NTIS AD# B059679 (June 1981).
85. 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 (18-20 May 1982).

86. 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).

87. 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).

88. 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).

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

90. 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).

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

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

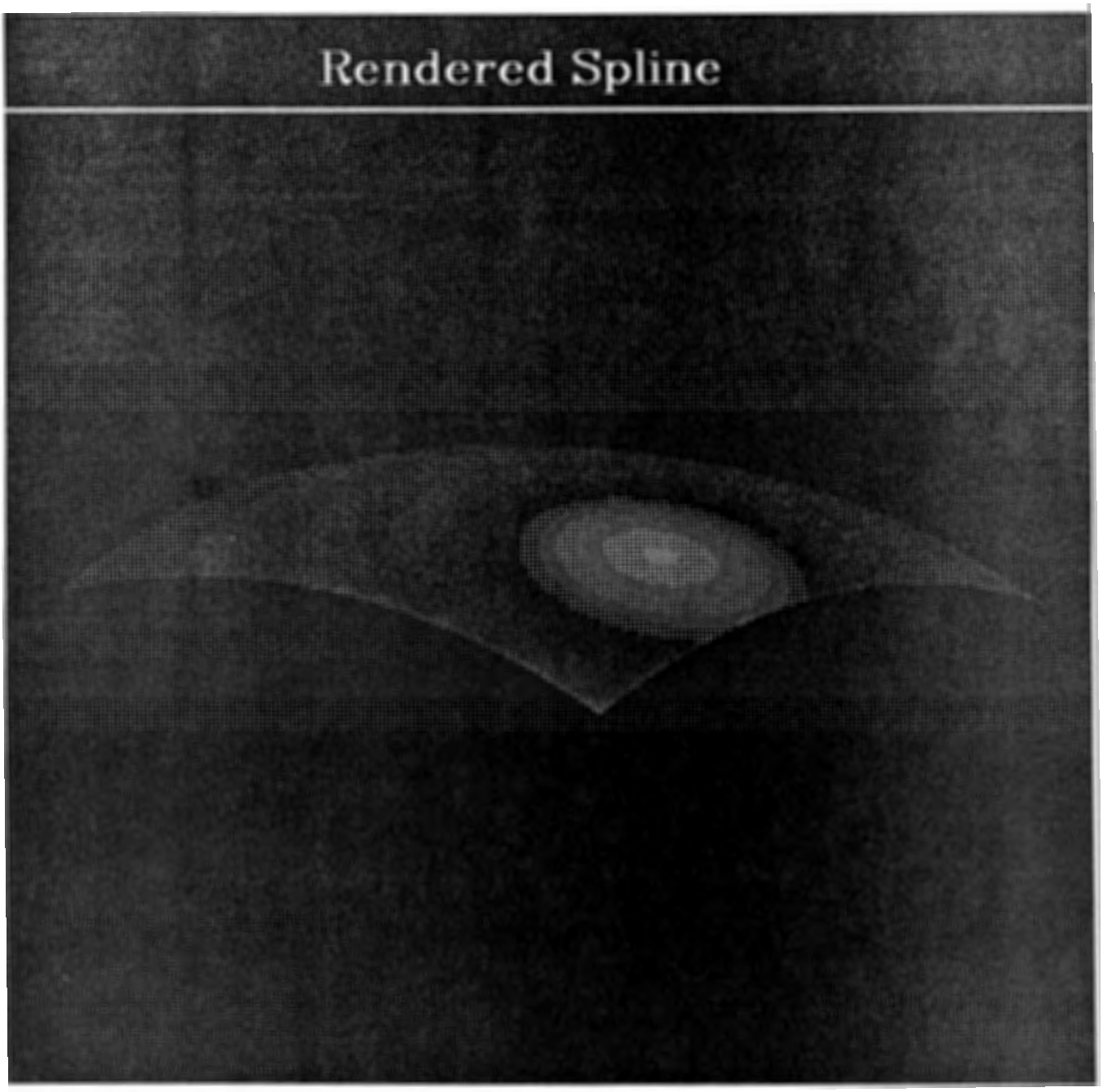
Plate Captions

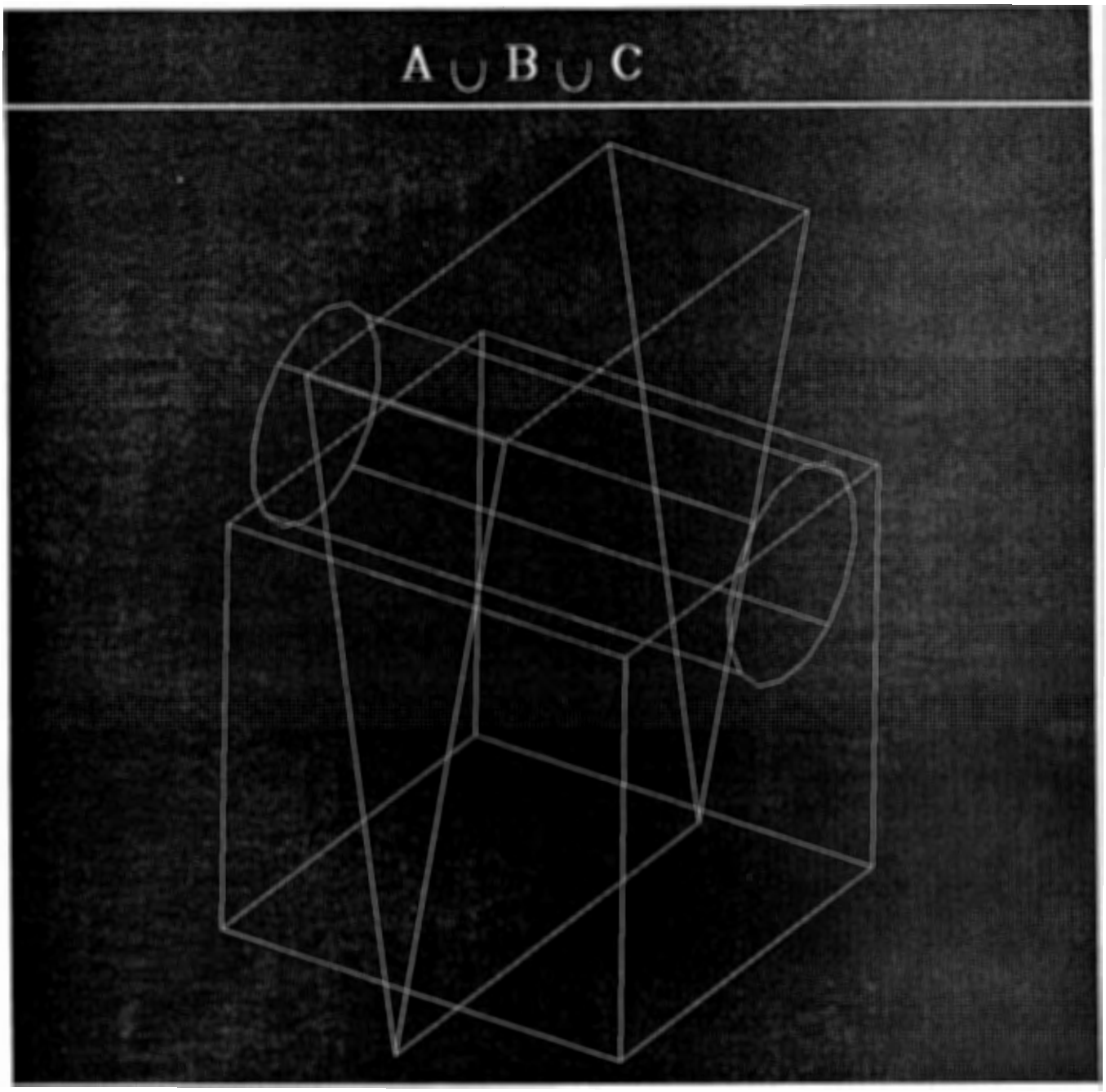
1.1 ComGeom Primitive Solids
1.2 Rendered Spline Surface
1.3 AuBuC Wireframe
1.4 AuBuC Rendering
1.5 A-B-C Wireframe
1.6 A-B-C Rendering
1.7 (A-B)+C Wireframe
1.8 (A-B)+C Rendering
1.9 C-A-B Wireframe
1.10 C-A-B Rendering
1.11 Early Rendering of Vehicle

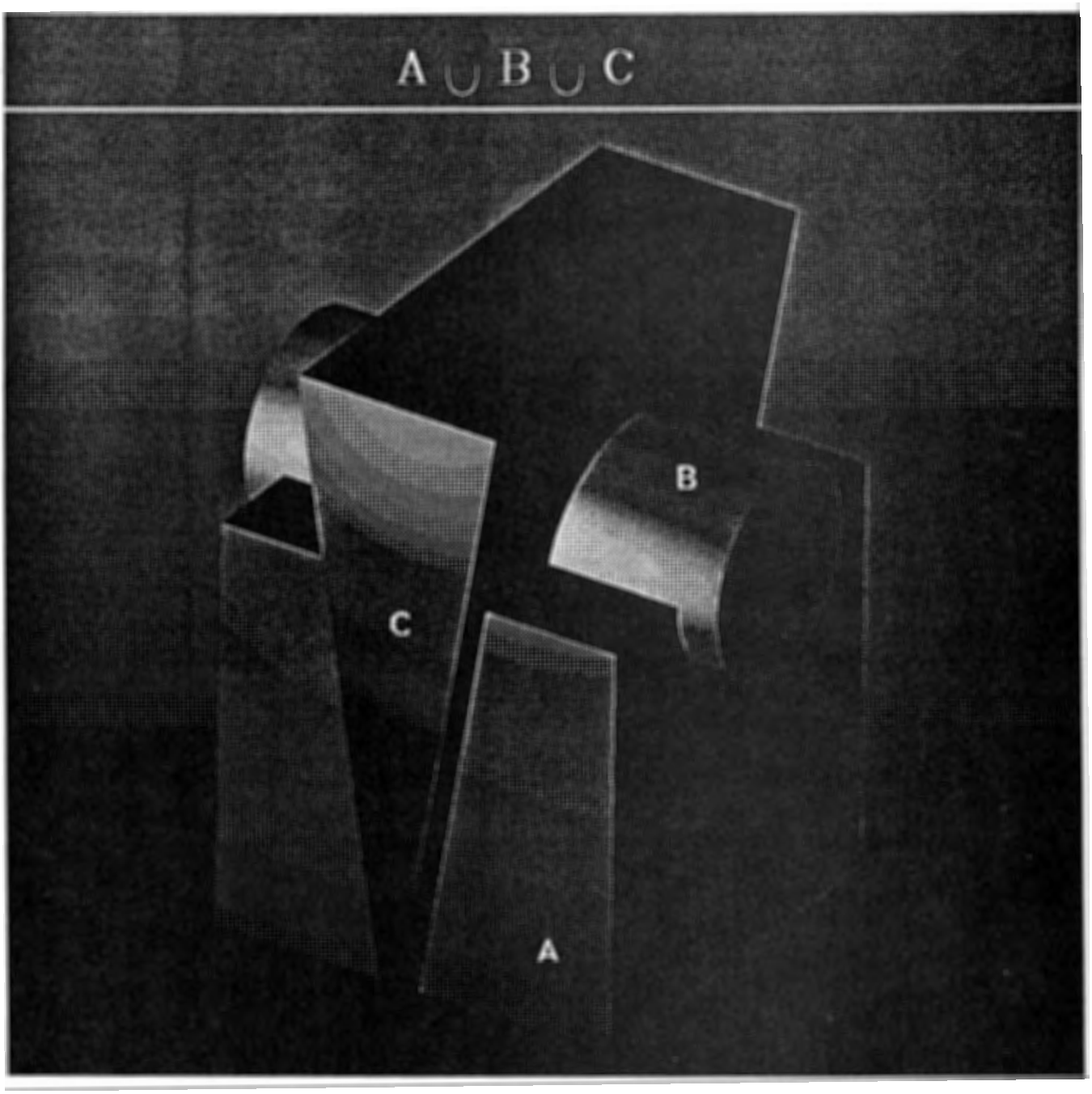
2.1 MGED Faceplate with First-level Menu
2.2 Engine Connecting Rod. Unevaluated and Evaluated Wireframes
2.3 Close-up of Connecting Rod Joint
2.4 ARB8 Before Editing
2.5 ARB8 After Editing Edge 1-4

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

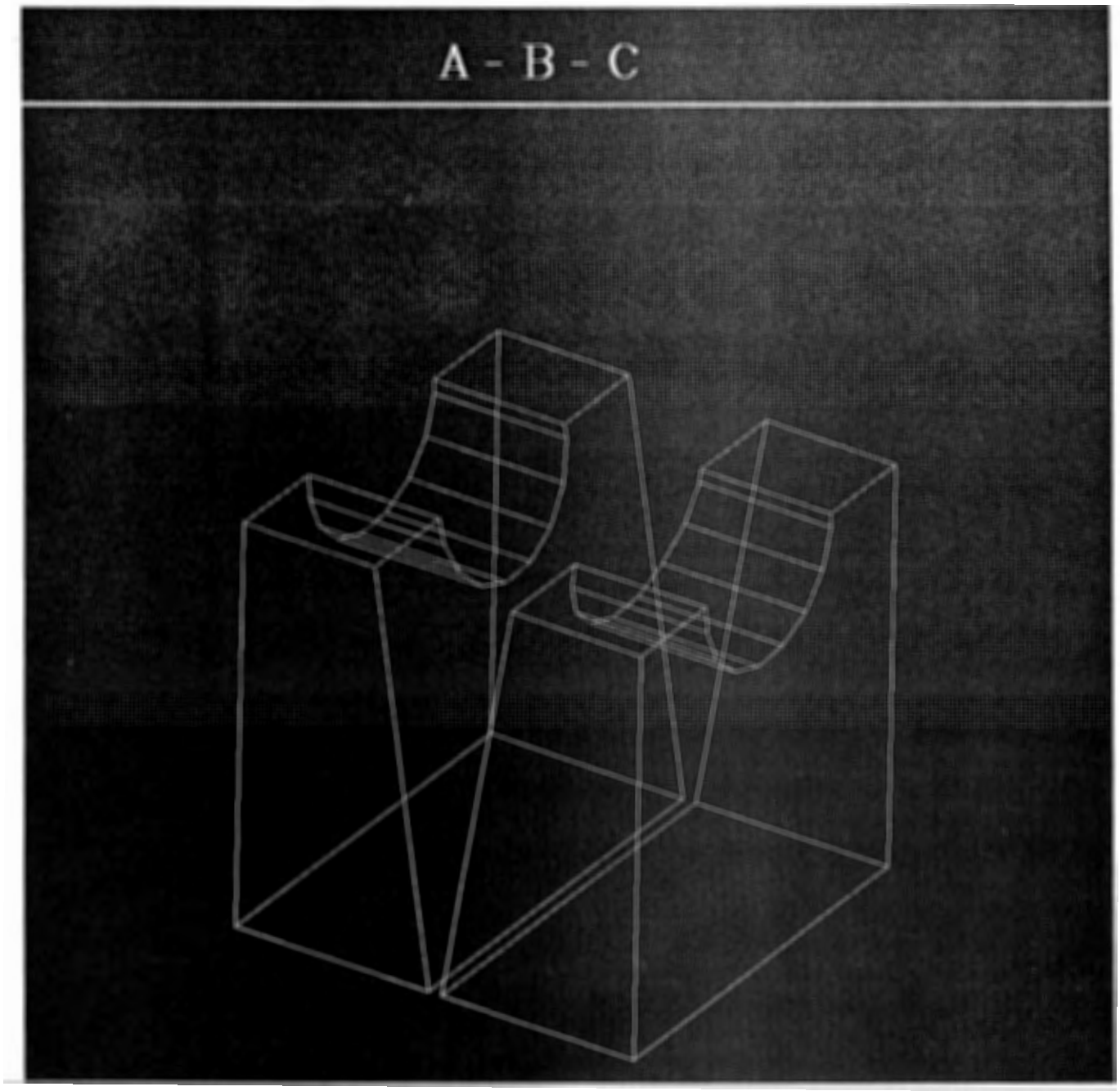
1.2

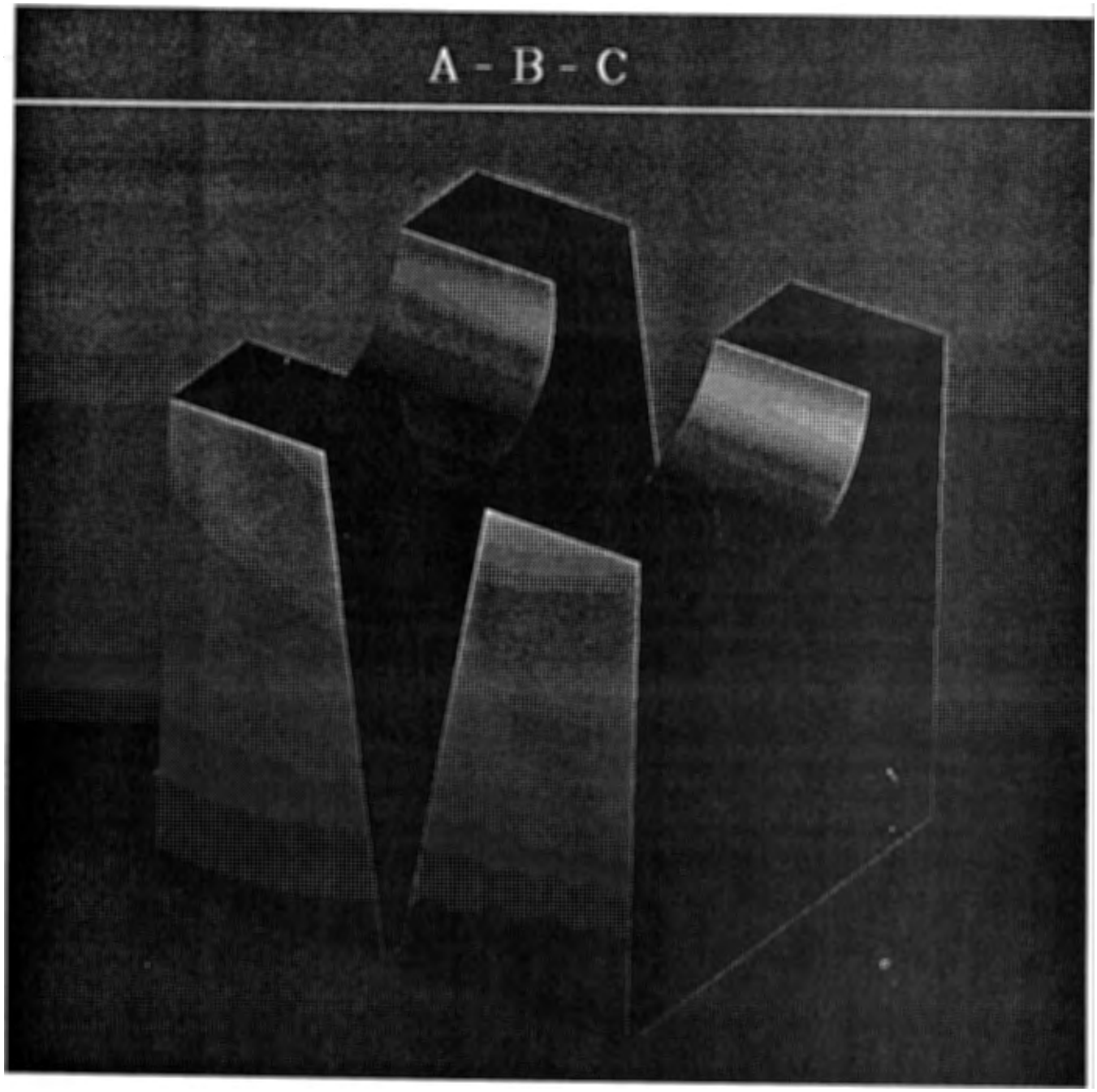


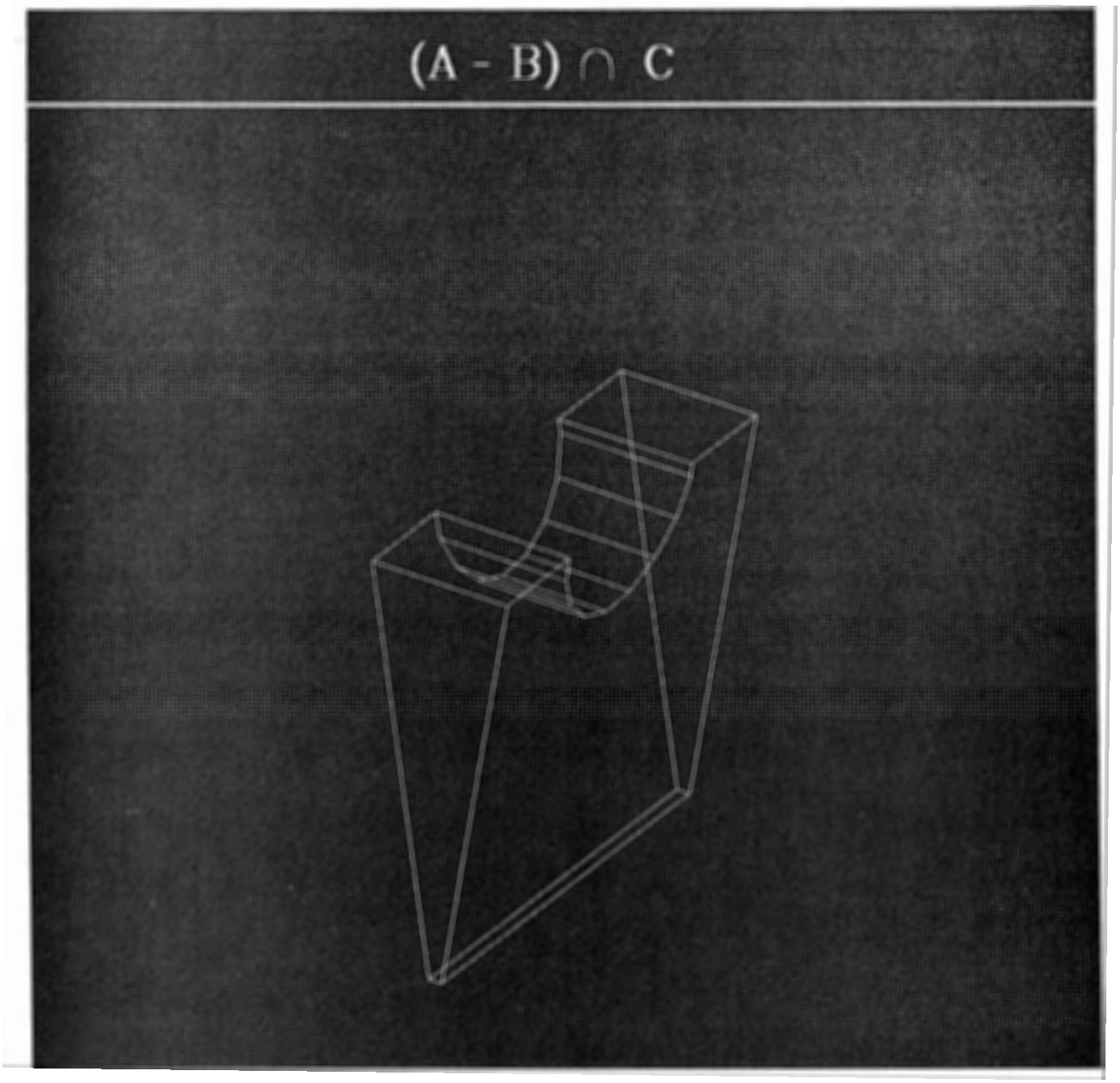

A ∪ B ∪ C


A ∪ B ∪ C
B
C
A

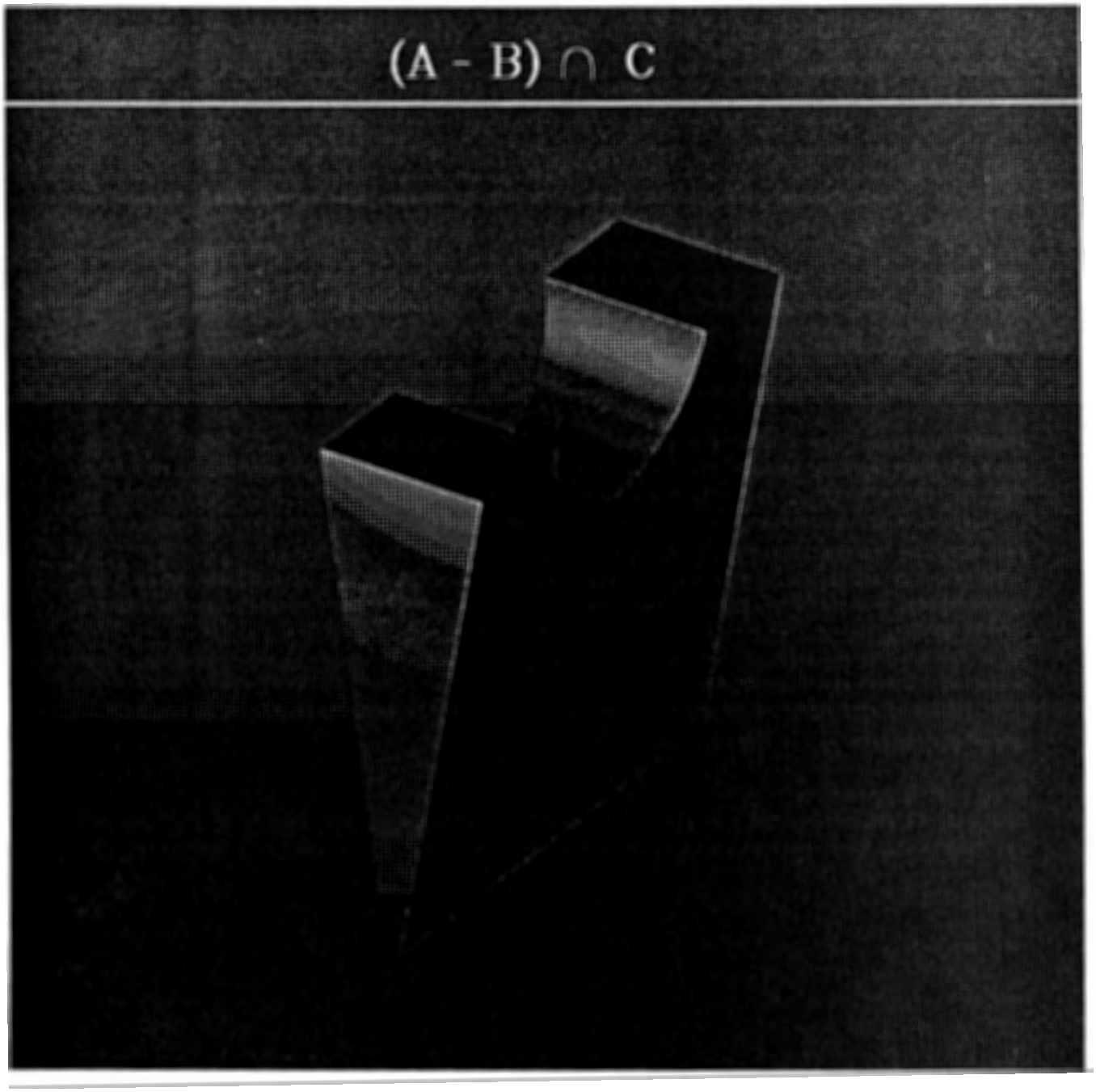
1.5



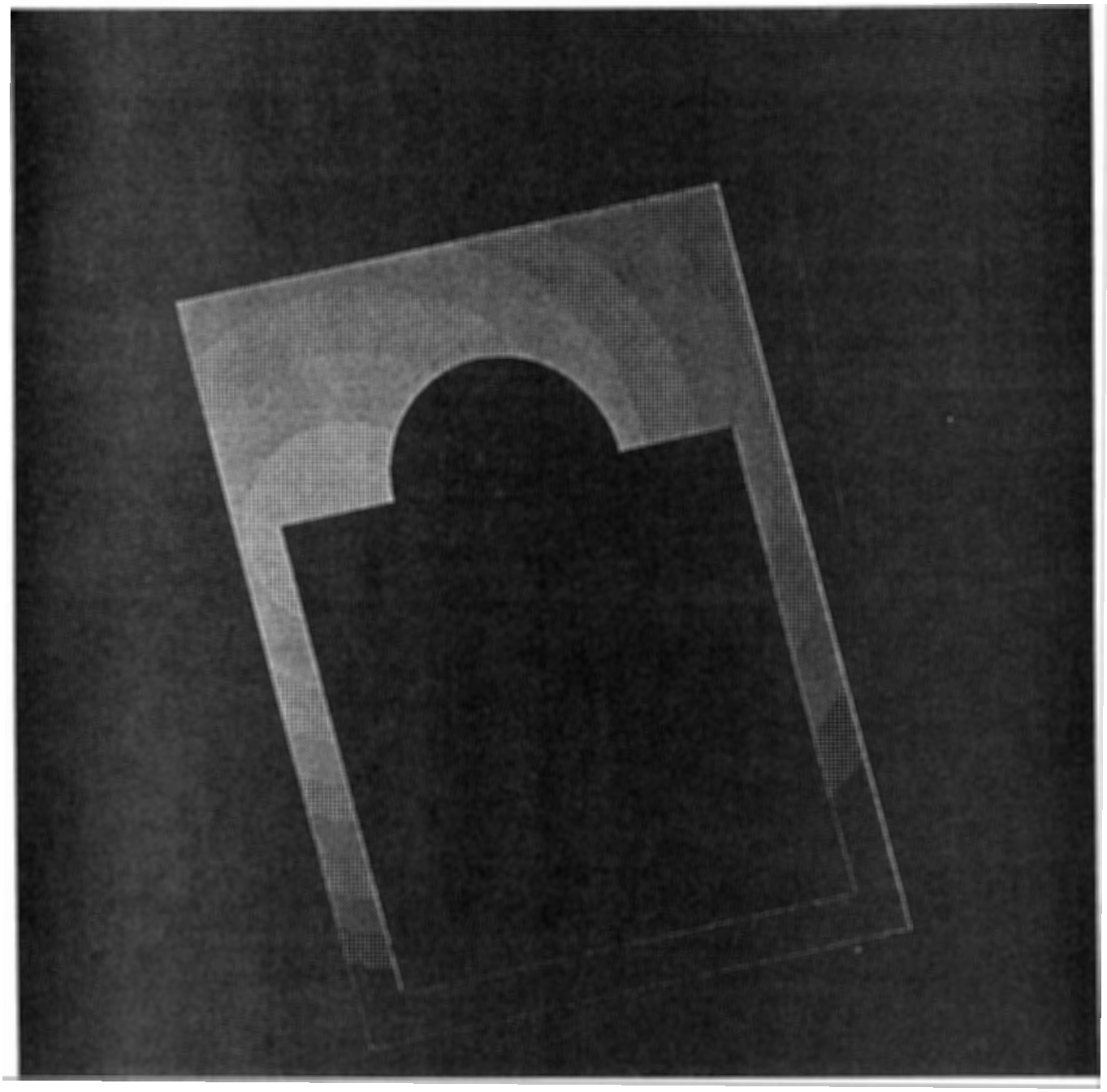

A - B - C


(A - B) ∩ C

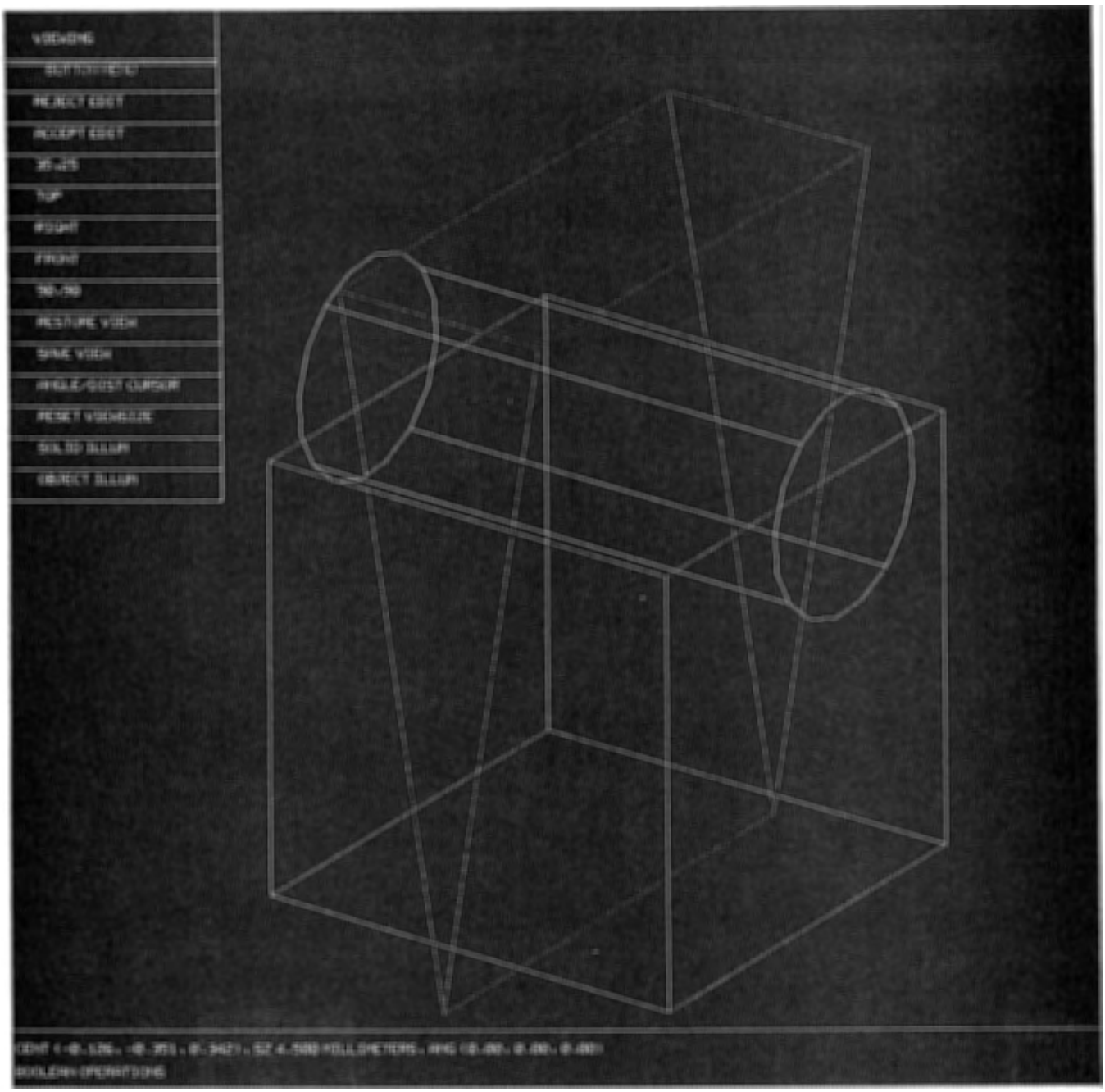
1.8



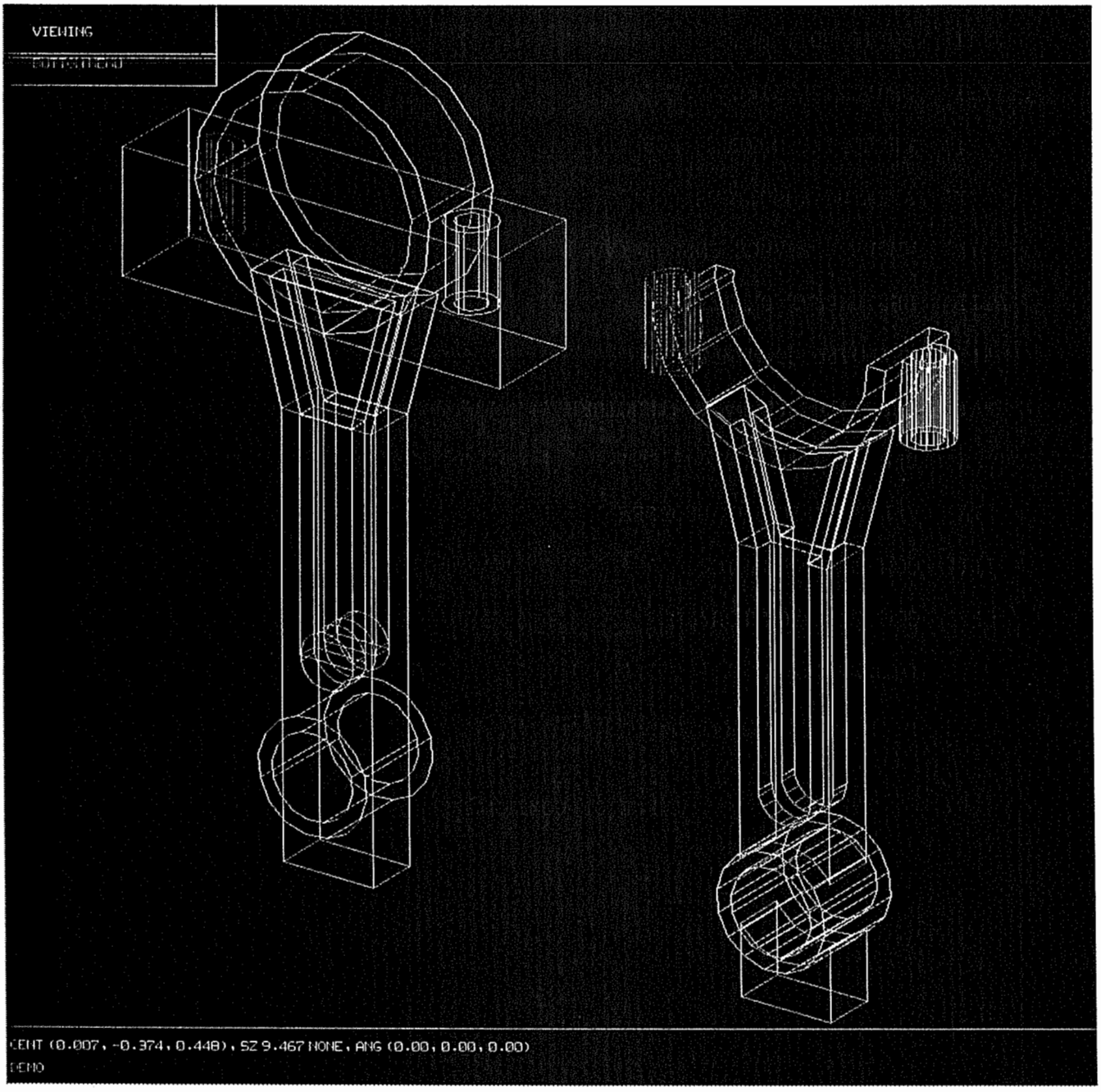
1.10



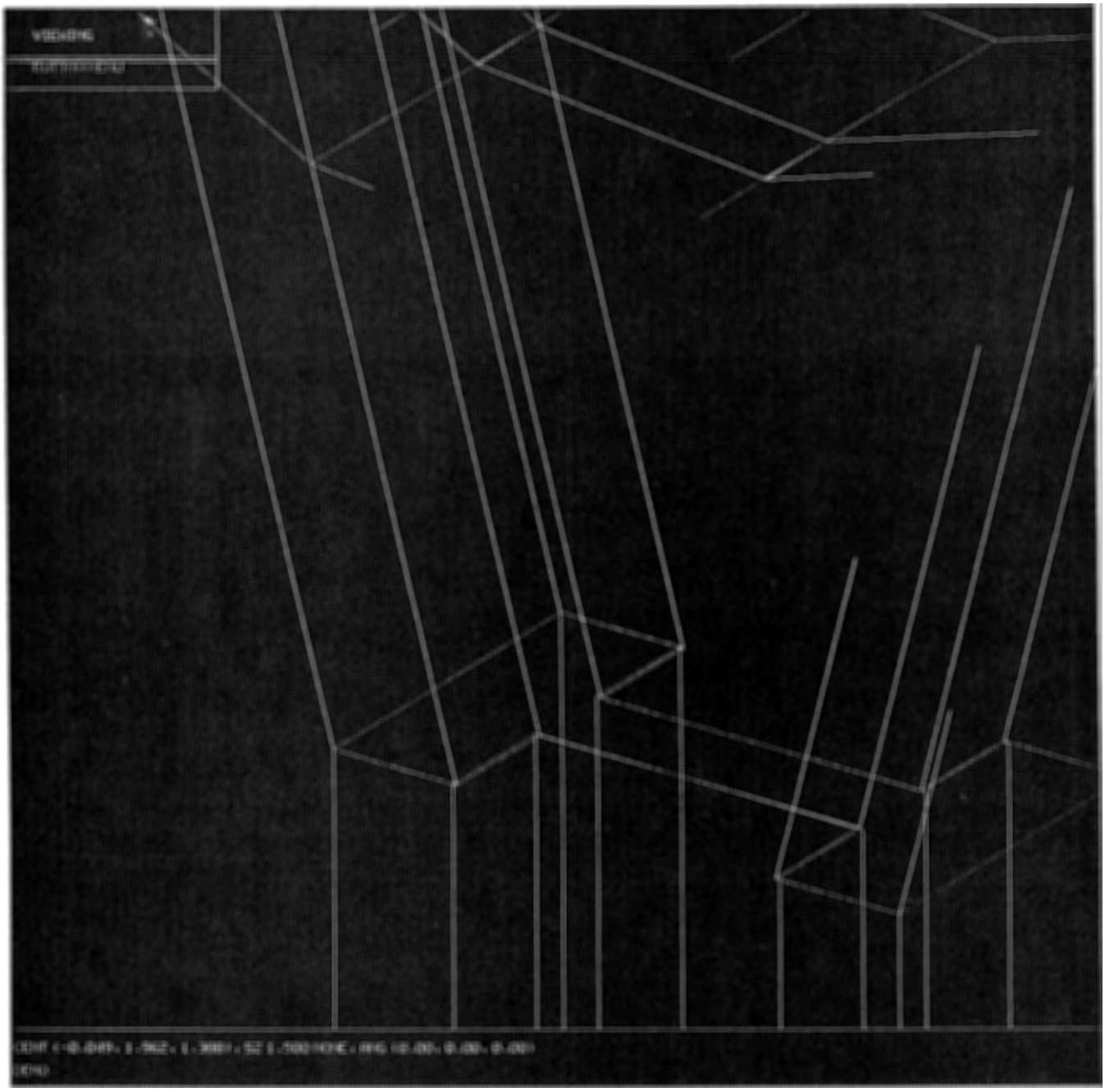
2.1

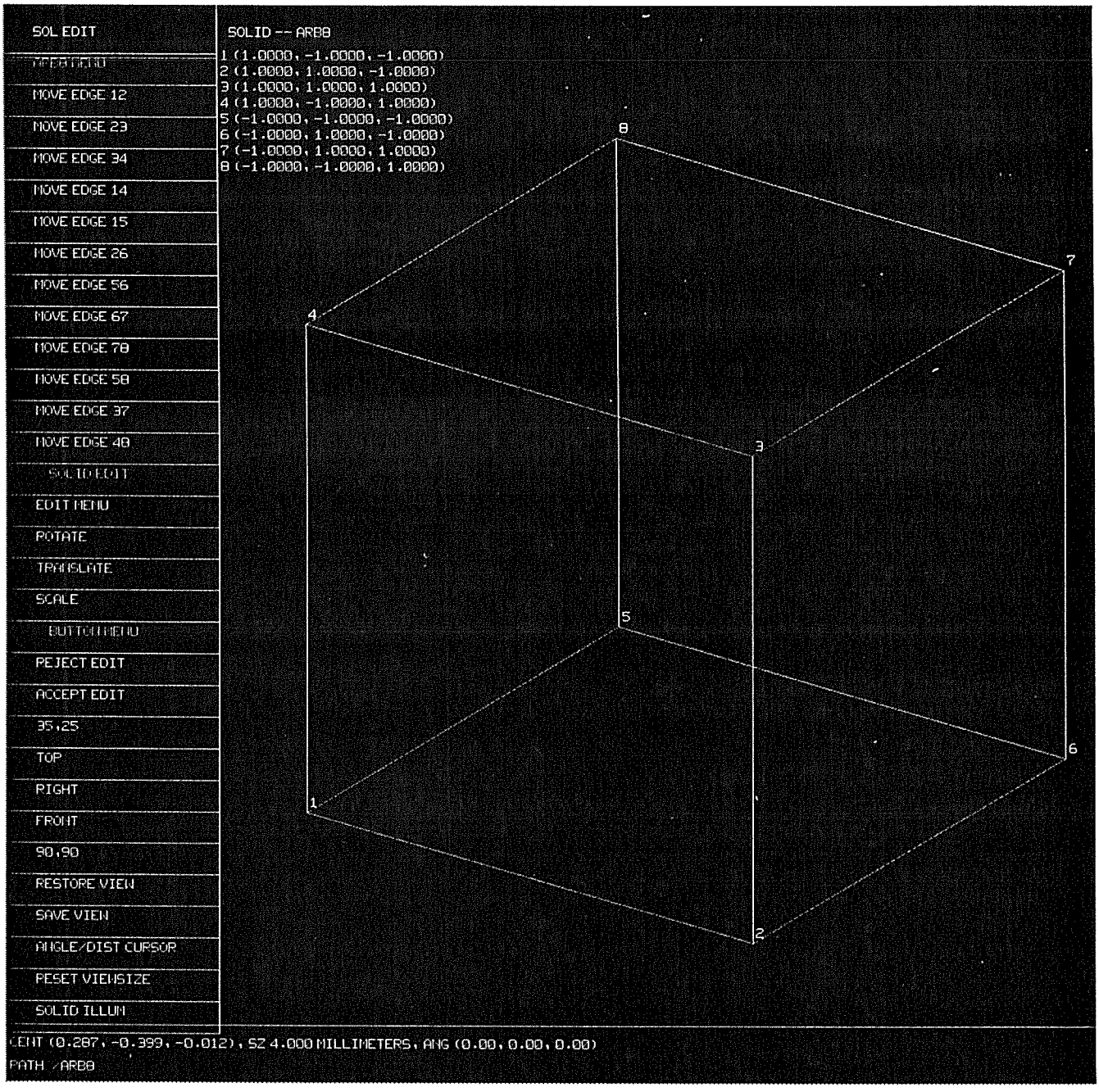

VIEWING
REJECT EDIT
ACCEPT EDIT
TOP
RIGHT
FRONT
RESTORE VIEW
SAVE VIEW

2.2

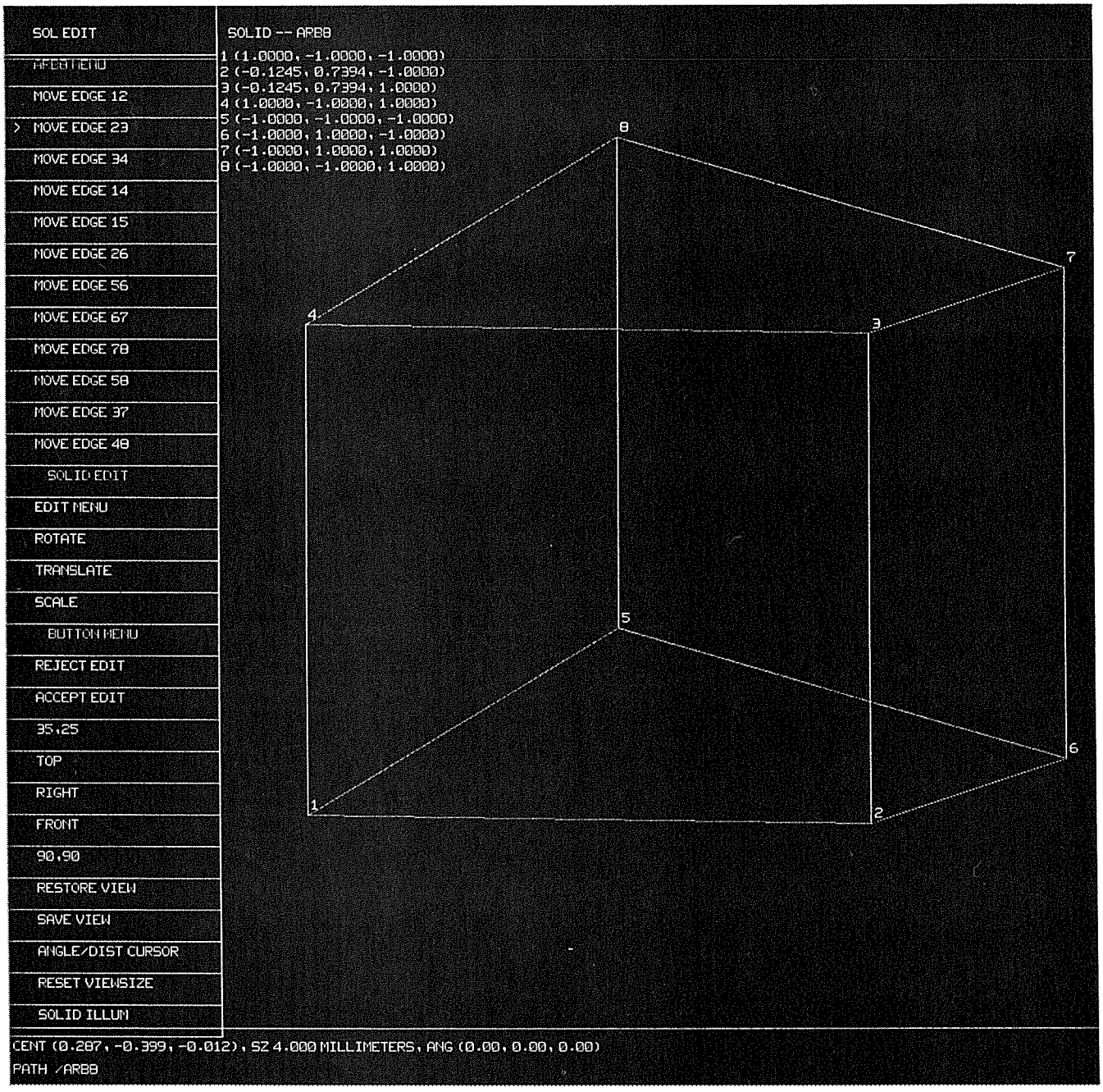


2.3



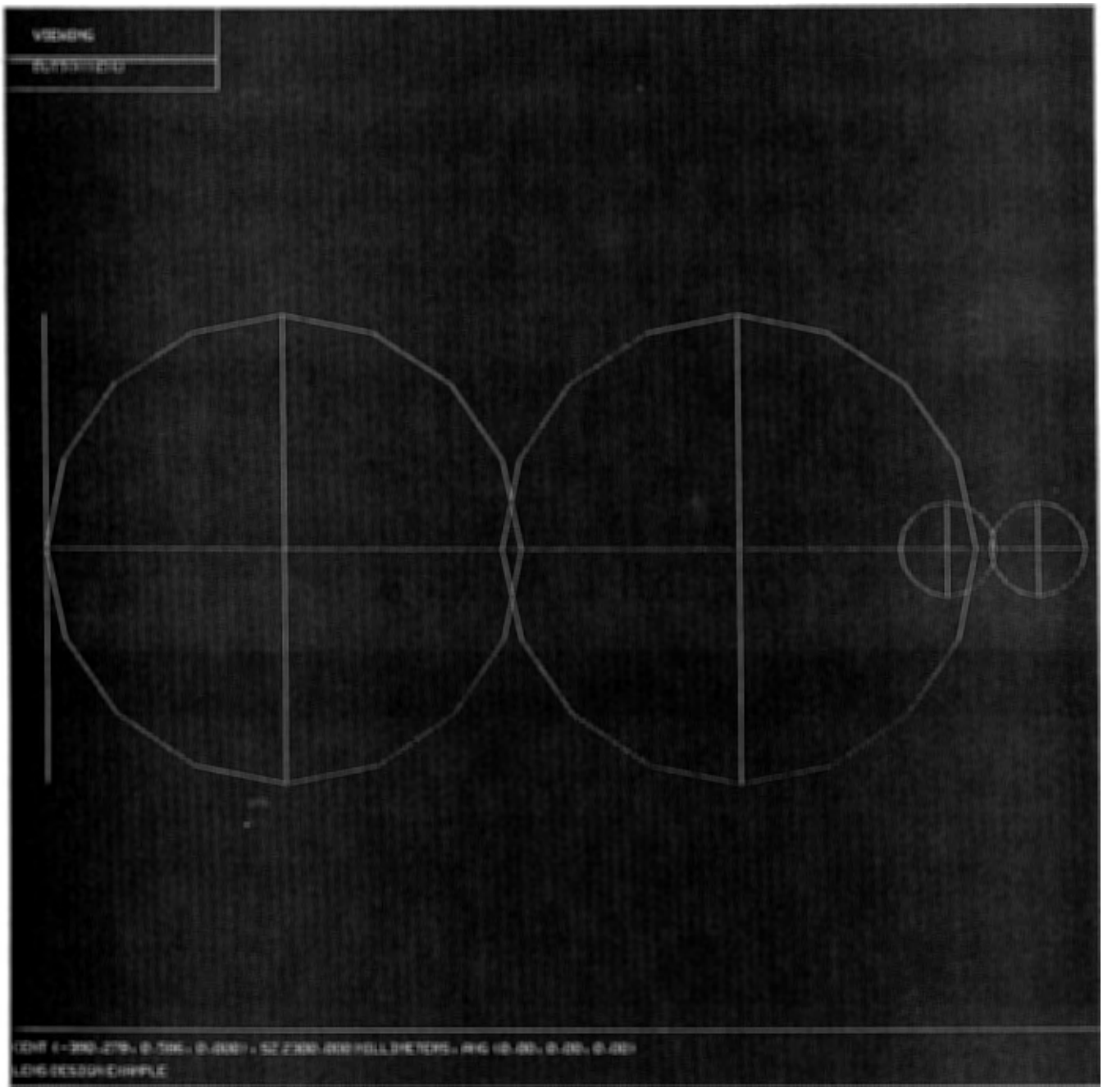

SOL EDIT
MOVE EDGE 12
MOVE EDGE 23
MOVE EDGE 34
MOVE EDGE 14
MOVE EDGE 15
MOVE EDGE 26
MOVE EDGE 56
MOVE EDGE 67
MOVE EDGE 78
MOVE EDGE 58
MOVE EDGE 37
MOVE EDGE 48
SOLID EDIT
EDIT MENU
ROTATE
TRANSLATE
SCALE
BUTTON MENU
REJECT EDIT
ACCEPT EDIT
35,25
TOP
RIGHT
FRONT
90,90
RESTORE VIEW
SAVE VIEW
ANGLE/DIST CURSOR
RESET VIEWSIZE
SOLID ILLUM
SOLID -- ARB8
1 (1.0000, -1.0000, -1.0000)
2 (1.0000, 1.0000, -1.0000)
3 (1.0000, 1.0000, 1.0000)
4 (1.0000, -1.0000, 1.0000)
5 (-1.0000, -1.0000, -1.0000)
6 (-1.0000, 1.0000, -1.0000)
7 (-1.0000, 1.0000, 1.0000)
8 (-1.0000, -1.0000, 1.0000)
8
7
4
3
5
6
1
2
CENT (0.287, -0.399, -0.012), SZ 4.000 MILLIMETERS, ANG (0.00, 0.00, 0.00)
PATH /ARB8

2.5

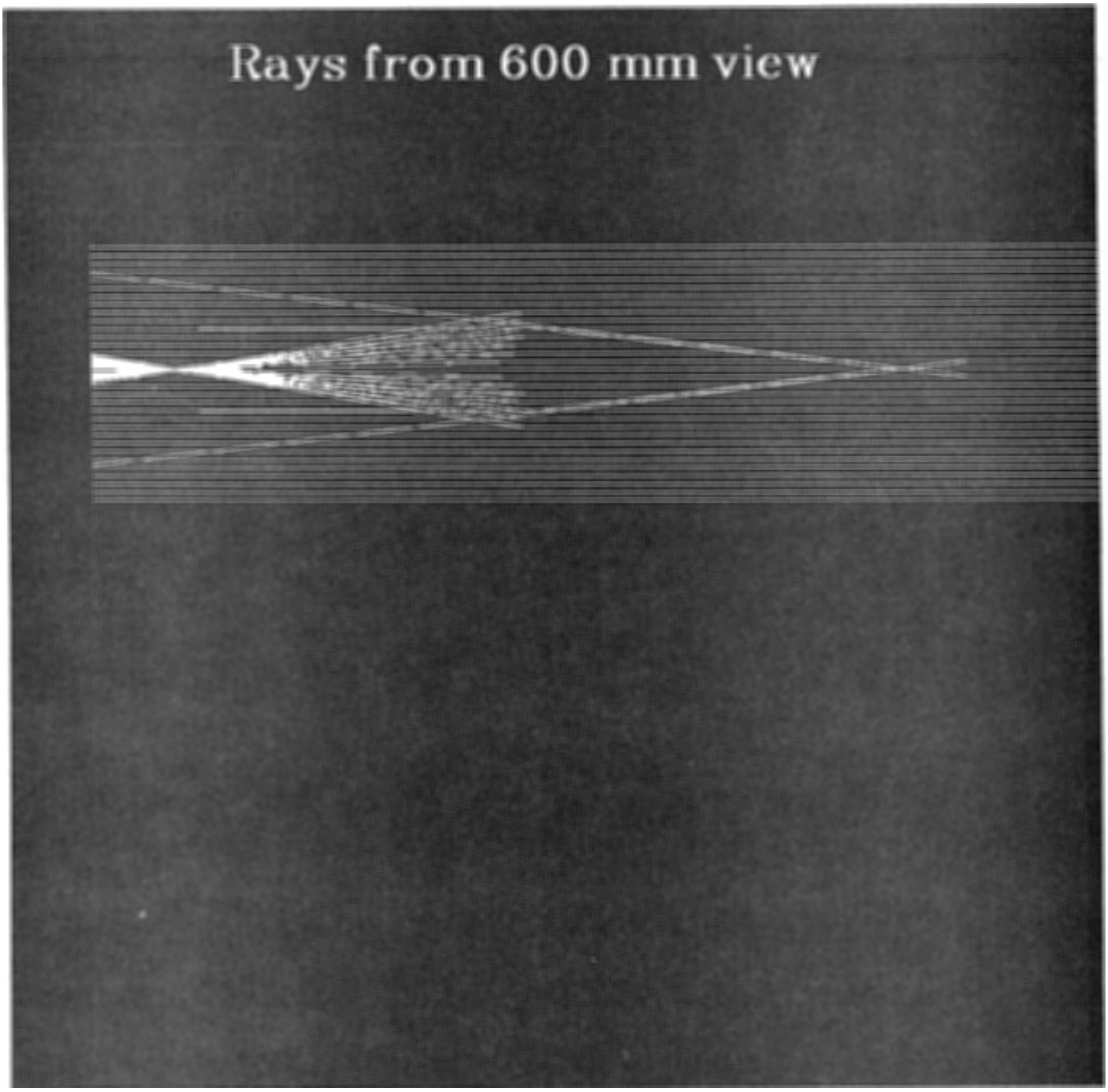

SOL EDIT
ARB8 MENU
MOVE EDGE 12
> MOVE EDGE 23
MOVE EDGE 34
MOVE EDGE 14
MOVE EDGE 15
MOVE EDGE 26
MOVE EDGE 56
MOVE EDGE 67
MOVE EDGE 78
MOVE EDGE 58
MOVE EDGE 37
MOVE EDGE 48
SOLID EDIT
EDIT MENU
ROTATE
TRANSLATE
SCALE
BUTTON MENU
REJECT EDIT
ACCEPT EDIT
35,25
TOP
RIGHT
FRONT
90,90
RESTORE VIEW
SAVE VIEW
ANGLE/DIST CURSOR
RESET VIEWSIZE
SOLID ILLUM
SOLID -- ARB8
1 (1.0000, -1.0000, -1.0000)
2 (-0.1245, 0.7394, -1.0000)
3 (-0.1245, 0.7394, 1.0000)
4 (1.0000, -1.0000, 1.0000)
5 (-1.0000, -1.0000, -1.0000)
6 (-1.0000, 1.0000, -1.0000)
7 (-1.0000, 1.0000, 1.0000)
8 (-1.0000, -1.0000, 1.0000)
8
7
4
3
5
6
1
2
CENT (0.287, -0.399, -0.012), SZ 4.000 MILLIMETERS, ANG (0.00, 0.00, 0.00)
PATH /ARB8

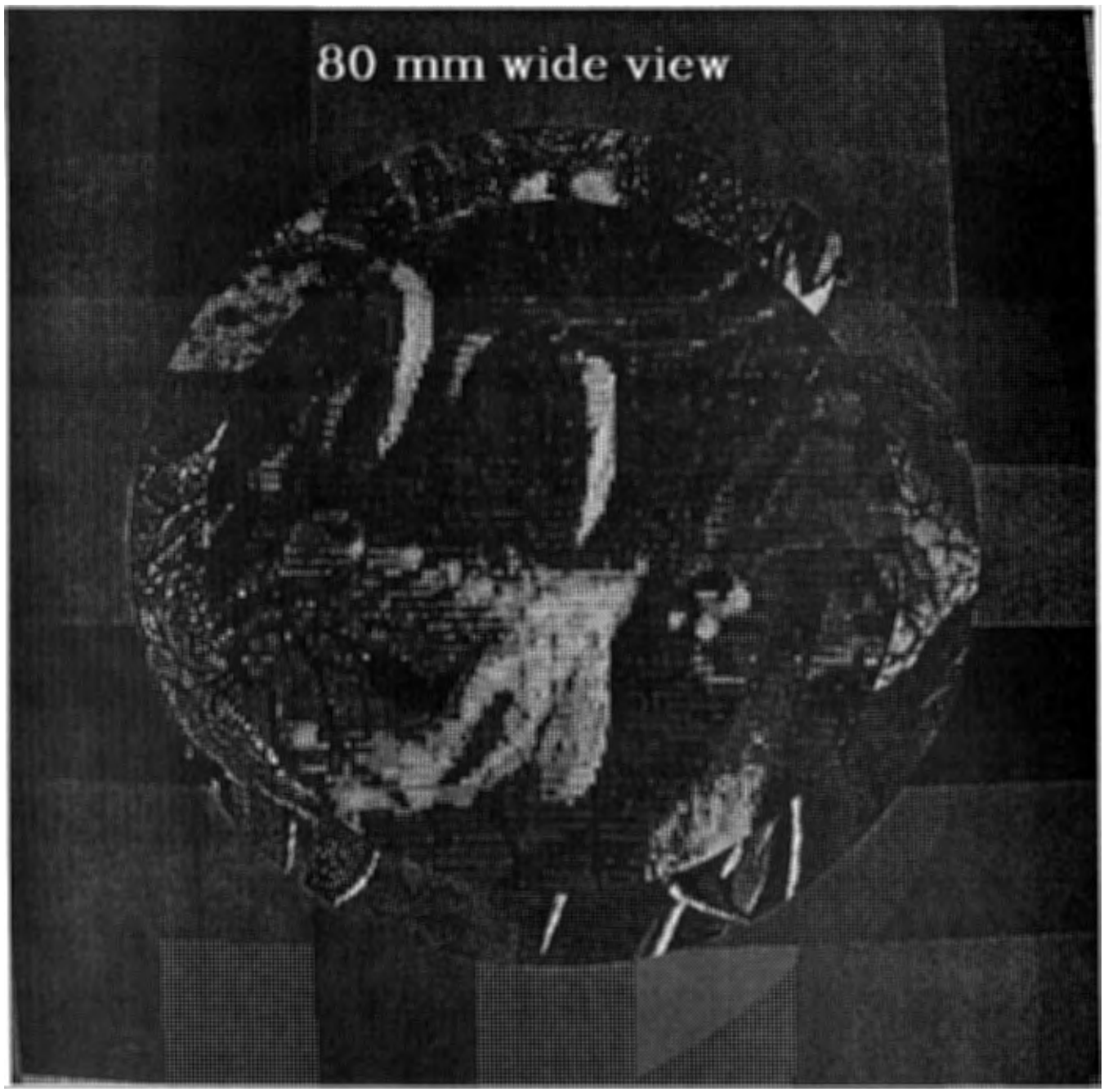
M48 with Spline-Solid Turret

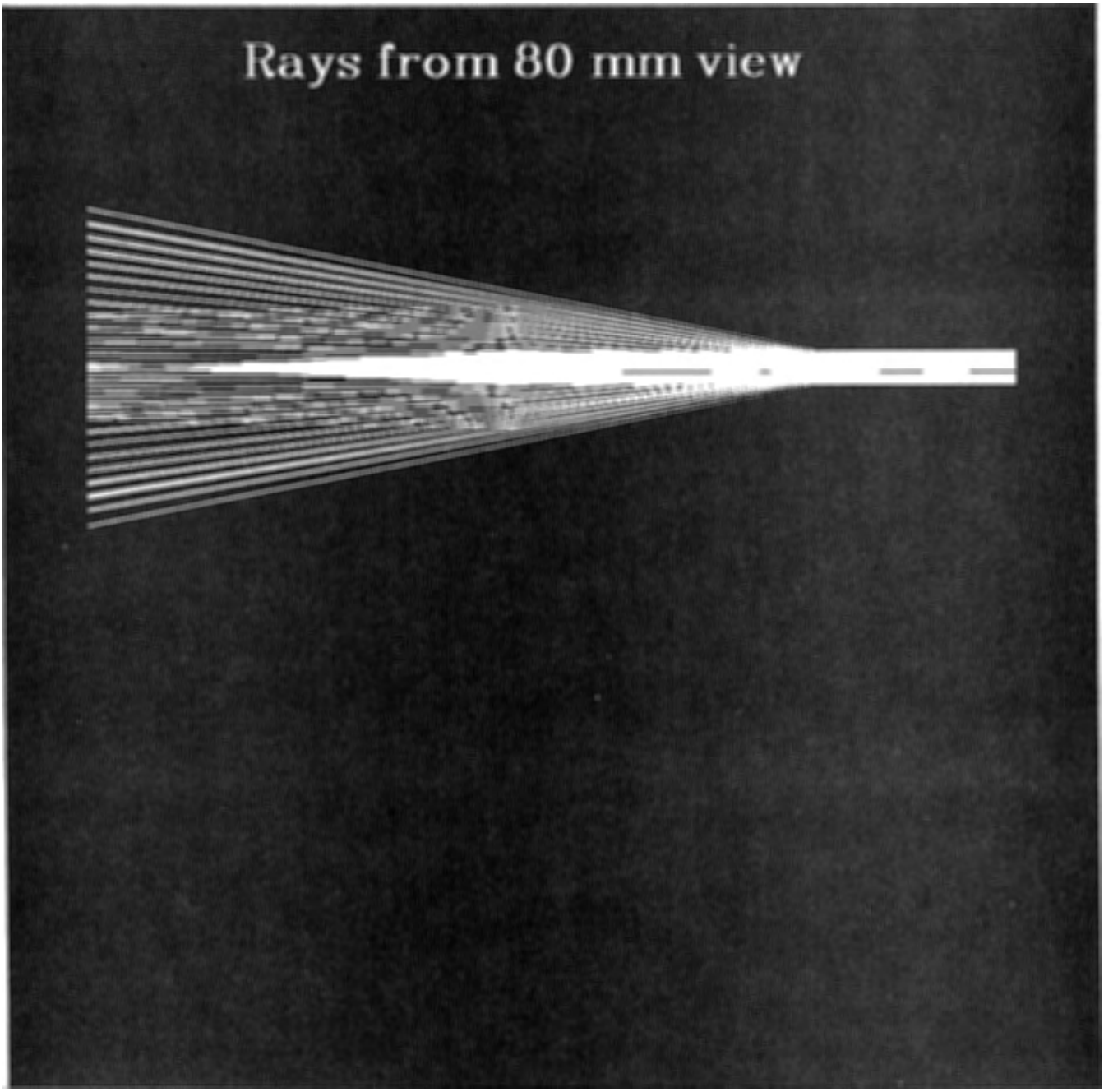





600 mm wide view


Rays from 600 mm view


80 mm wide view


Rays from 80 mm view

4.

GED: An Interactive Solid Modeling System for Vulnerability Assessments

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TECHNICAL REPORT ARBRL-TR-02480	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* GED: AN INTERACTIVE SOLID MODELING SYSTEM FOR VULNERABILITY ASSESSMENTS		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Michael J. Muuss, Gary S. Moss Keith A. Applin, Earl P. Weaver J. Robert Suckling, Charles A. Stanley		8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Ballistic Research Laboratory ATTN: DRDAR-BLV Aberdeen Proving Ground, MD 21005		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1L162618AH80
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Armament Research & Development Command U.S. Army Ballistic Research Laboratory (DRDAR-BL) Aberdeen Proving Ground, MD 21005		12. REPORT DATE March 1983 13. NUMBER OF PAGES 46
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Target Descriptions
Interactive
Computer Graphics
Combinatorial Geometry

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

For many years the production and modification of target descriptions, no matter what method used, has been a slow, labor-intensive procedure. In an effort to improve the response time of target descriptions, the Ballistic Research Laboratory (BRL) has developed an interactive solid modeling system. This system is called the Graphics Editor (GED) and is designed to replace the traditional manual method of production and modification of target descriptions. GED presently applies to the combinatorial geometry (COMGEOM) solid modeling (cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

technique. Using GED, the target descriptions are interactively viewed, modified, and constructed with immediate visual feedback at each step. The GED system virtually eliminates explicit numerical input and opens a new dimension in the target description process.

TABLE OF CONTENTS

		Page
	LIST OF ILLUSTRATIONS	5
I.	INTRODUCTION	7
II.	COMGEOM BACKGROUND	7
III.	COMPUTER GRAPHICS	9
IV.	APPROACH TO THE PROBLEM	9
V.	GED OVERVIEW	11
VI.	GED INTERNAL DATA STRUCTURE	11
VII.	KEYBOARD COMMANDS	12
VIII.	PERIPHERAL DEVICES	22
IX.	ANGLE DISTANCE CURSOR (ADC)	22
X.	VIEWING FUNCTIONS	26
XI.	SELECTING OBJECTS FOR EDITING	27
XII.	OBJECT EDITING	28
XIII.	SOLID EDITING	29
	A. ARB Parameter Editing	32
	B. TGC Parameter Editing	34
	C. ELLG Parameter Editing	36
	D. TOR Parameter Editing	36
XIV.	CONCLUSION	38
	DISTRIBUTION LIST	41

LIST OF ILLUSTRATIONS

Figure		Page
1.	Logic Operations Used to Combine Solids.	10
2.	Sample Hierarchy of Object "heround"	13
3.	Tasks Assigned to the Function Switch Buttons.	23
4.	Functions Assigned to the Control Knobs.	24
5.	The Angle Distance Cursor.	25
6.	Typical Generalized ARBs	33
7.	Typical TGCs	35
8.	Typical ELLGs.	37
9.	TOR Parameters	39

I. INTRODUCTION

As the Army's lead laboratory for vulnerability technology, the Ballistic Research Laboratory (BRL) constantly performs vulnerability analyses for a wide variety of military systems. Vital to these vulnerability studies are three-dimensional computer models of the physical characteristics of these systems. These computer models are generally called target descriptions. Since the mid 1960's, BRL has used a solid modeling technique called combinatorial geometry (COMGEOM) to model targets. The COMGEOM technique uses logic operations to combine basic geometric shapes or primitives to produce the three-dimensional target description. The COMGEOM target descriptions are processed by the Geometric Information For Targets (GIFT)[1,2] code for use in follow-on vulnerability assessment codes.

Target descriptions are basically large collections of numerical data which traditionally have been handled manually in a batch environment. Hence, the production and modification of target descriptions has been a slow, labor-intensive process. In 1980, BRL initiated an effort to improve the response time of the target description process by applying interactive computer graphics techniques. As a result of this work unit, BRL has developed the Graphics EDitor (GED), an interactive solid modeling system based on the COMGEOM technique. Using GED an analyst can view, build, and modify target descriptions interactively by manipulating the graphical representation, thus providing immediate visual feedback on a vector display device. The GED system is designed to replace the manual method of production and modification of target descriptions.

This report is intended to serve as a user manual for the GED system. The process of viewing and editing a description using GED is covered in detail. The internal data structure is also covered, as it is an important part in the overall design of the GED system.

II. COMGEOM BACKGROUND

Since the GED system is presently based on the COMGEOM solid modeling technique, a brief overview of the COMGEOM technique is required to effectively use GED. For more detailed information on the COMGEOM technique see References 1 and 2.

The COMGEOM technique utilizes two basic entities - a solid and a region. A solid is defined as one of fifteen basic geometric shapes or primitives. Table I contains a list of these fifteen solids. The individual parameters of each solid define the solid's location, size,

---

[1] Lawrence W. Bain, Mathew J. Reisinger, "The GIFT Code User Manual; Vol I, Introduction and Input Requirements," BRL Report No. 1802, July 1975 (AD# B0060371)

[2] Gary G. Kuehl, Lawrence W. Bain, Mathew J. Reisinger, "The GIFT Code User Manual; Vol II, the Output Options," ARRADCOM Technical Report No. ARBRL-TR-02189, September 1979 (AD# A078364)

Table I. COMGEOM SOLIDS

Symbol	Name
ARS	Triangular Surfaced Polyhedron
ARB	Arbitrary Convex Polyhedron
RPP	Rectangular Parallelepiped
BOX	Box
RAW	Right Angle Wedge
ELLG	General Ellipsoid
ELL	Ellipsoid of Revolution
SPH	Sphere
TGC	Truncated General Cone
RCC	Right Circular Cylinder
REC	Right Elliptical Cylinder
TRC	Truncated Right Cylinder
TEC	Truncated Elliptical Cylinder
TOR	Torus
HAF	Half Space

and orientation. A region is a combination of one or more solids and is defined as the volume occupied by the resulting combination of solids. Solids are combined into regions using any of three logic operations: union(OR), intersection(+), or difference(-). The union of two solids is defined as the volume in either of the solids. The difference of two solids is defined as the volume of the first solid minus the volume of the second solid. The intersection of two solids is defined as the volume common to both solids. Figure 1 presents a graphical representation of these operations. Any number of solids may be combined to produce a region. As far as the COMGEOM technique is concerned, only a region can represent a component of the target. The solids are only building blocks to be combined into regions. Since regions represent the components of the target system, they are further identified by code numbers. These code numbers either identify the region as a target component (nonzero item code) or as air (nonzero air code). Any volume not defined as a region is assumed to be "universal air" and is given an air code of "01." If it is necessary to distinguish between universal "01" air and any other kind of air, then that volume must be defined as a region and given an air code other than "01." Normally, regions cannot occupy the same volume (overlap), but regions identified with air codes can overlap with any region identified as a component (i.e., one that has a nonzero item code). Regions identified with different air codes,however, can not overlap.

III. COMPUTER GRAPHICS

Computer graphics is one of the fastest growing fields in the computer industry. Computer graphics has applications in many diverse areas, from electronic games to medicine, from cartoons to the space industry. Just what is interactive computer graphics and why is it so versatile? Human visual perception is quite keen and communications with a computer are generally faster and easier with images or displays rather than with numbers. Furthermore, by driving the display image with the computer, one can actually communicate with the computer through the display itself. This technology is called interactive computer graphics. The user converses with the computer through the display using devices such as light pens, data tablets, function switches, and control dials. The response of the computer is immediately reflected on the display. Thus,a fast channel of communication is available between person and machine.

IV. APPROACH TO THE PROBLEM

To speed up the model description process, a faster, more efficient method of manipulating the large masses of data involved in a target description is needed. Interactive computer graphics seems to provide the capability of handling such data in an efficient, relatively fast manner. The key to the whole situation is SOFTWARE - the missing link between the target description problem and a possible solution.

In view of this fact, existing software graphics packages were evaluated with respect to their utility in the target description

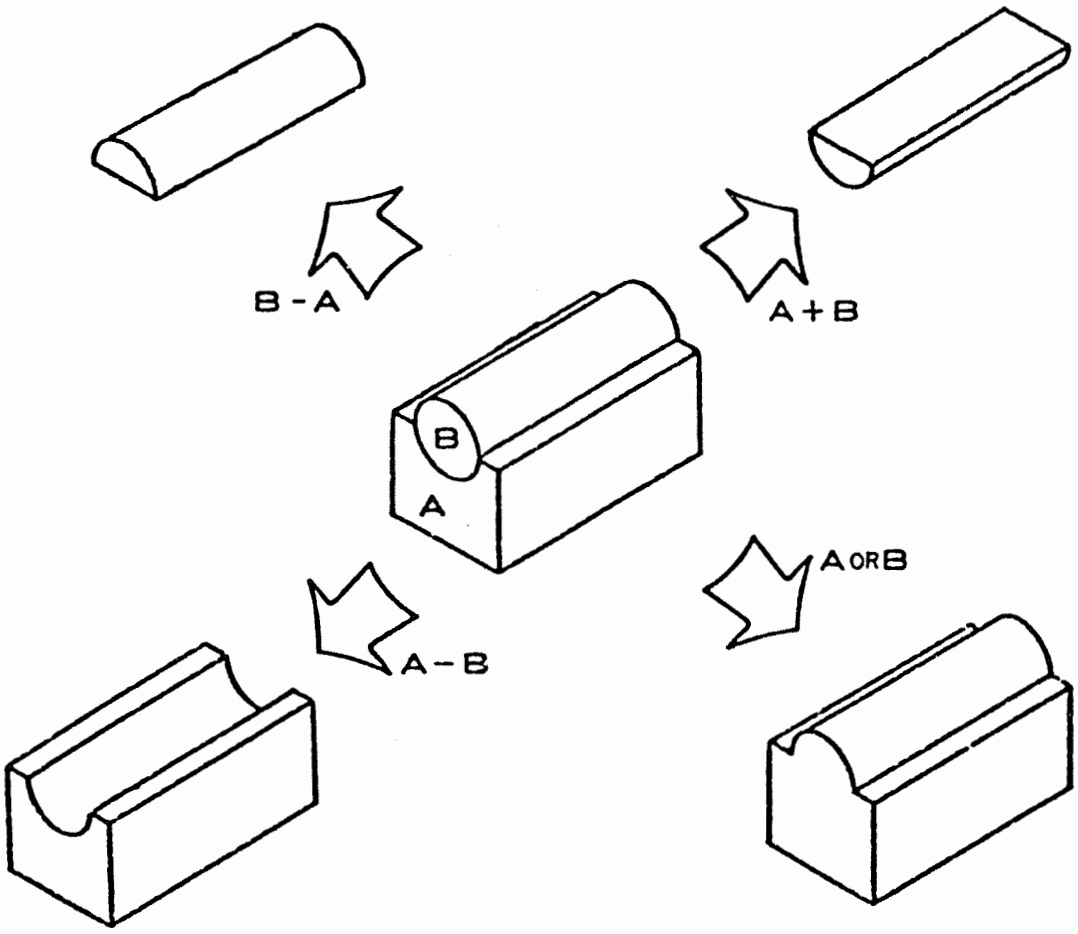


Figure 1. Logic Operations Used to Combine Solids

process. Quite an exhaustive search of commercially available systems was conducted, and none was found which completely satisfied our requirements. A study was then initiated to examine the feasibility of producing the required graphics software in-house. The result of this study was a preliminary version of GED which, using existing hardware, quite convincingly demonstrated the feasibility of such an undertaking.[3] It was then decided to develop GED into a full production code and to acquire the hardware necessary for full implementation of the system. The production level GED code has been used since January 1982 to build target descriptions interactively at BRL.

V. GED OVERVIEW

The main function of the GED system is to edit target descriptions with immediate visual feedback. Hence, GED produces images both suitable for editing and capable of being displayed in real time. When an object is displayed, the solids combined to model the object are drawn. No attempt is made to produce edge representations of regions or to remove hidden lines. At the present, such an undertaking would take away from the "real time" displays now enjoyed.

The GED is a modular code written in the C programming language which presently runs under the BRL UNIX Networking System.[4] The work stations presently used are Vector General (VG) 3DI vector refresh systems driven by DEC PDP-11/34 or PDP-11/70 computers. Peripheral devices include a joystick, a pen and data tablet, function switches, and control dials. These peripheral devices and functions of each are discussed in later sections.

Overall control of GED is maintained via the keyboard. The keyboard commands perform general housekeeping functions and allow one to control what is displayed, create/modify combinations, list solid parameters, etc. The keyboard commands are discussed in a later section.

VI. GED INTERNAL DATA STRUCTURE

One of the critical aspects of a graphics software package is its internal data structure. Since the target description problem deals with such large volumes of data, the importance of the data structure here is emphasized. Thus, it is felt that an introduction to the internal data structure of GED is important for any potential user.

---

[3] Michael John Muuss, Earl P. Weaver, "Interactive Construction of COMGEOM Targets - A Feasibility Study," report to be published.

[4] Michael John Muuss, "BRLNET - The Implementation of a Multi-processor UNIX System," a final report of work done under ARO Project Number 1127, August 27, 1980.

The GED internal data structure is hierarchical in nature with each node or position in the hierarchy being occupied by an OBJECT. An OBJECT is the GED basic data unit and is defined as either a SOLID or a COMBINATION. A SOLID is one of the generalized COMGEOM primitive types while a COMBINATION is a group of objects. Each member object of a combination has a transformation associated with it. Any object not at the top of an hierarchy is referenced by (is a member of) a higher combination, and each such reference has an associated transformation. The bottom object of every hierarchy path is a solid. This hierarchical data structure allows actual subsystems of a target to be grouped together and edited as a unit without worrying about individual parts. To display any object, each path in the object's hierarchy is traveled to its bottom position where a solid is located. The transformations are accumulated for each path traversed and then applied to the solid at the bottom of the path. The solid is then drawn. The hierarchy is traversed recursively until all objects are drawn. Let's use a high explosive projectile as an example. Figure 2 depicts the hierarchy involved. If one were to display the combination "heround," six solids would be drawn on the screen. Table II lists each solid name, the transformation applied to the solid's parameters, and the path used to reach the solid.

One very useful type of combination is the INSTANCE. An instance is a referral to an object (solid or combination) without the object itself being duplicated. To produce an instance of an object, a combination is created with the object being instanced as a member object. As with any other combination, this member has an associated transformation, hence, it can be edited as a unit. The instance is most useful when an object appears many times in a description. The desired number of instances are made and positioned in the description. Any future changes in the object require that only the original (prototype) be changed. The modifications of the prototype object will then be automatically reflected in all the instances of that object.

VII. KEYBOARD COMMANDS

To execute GED, one simply types "ged file," where "file" is the data file to be edited. The GED keyboard commands are used to maintain overall control of the system and to perform general housekeeping functions. Each command is invoked by entering a single character followed by other parameters (if necessary) separated by spaces. In most cases, the response on the screen will be the word "done" once the command is completed. This response is not printed for the commands where the results of the command are readily visible on the screen.

In the following pages, the GED keyboard commands will be presented and discussed. Examples will be given in appropriate cases. Table III, at the end of this section, presents a summary of the GED keyboard commands and the function of each.

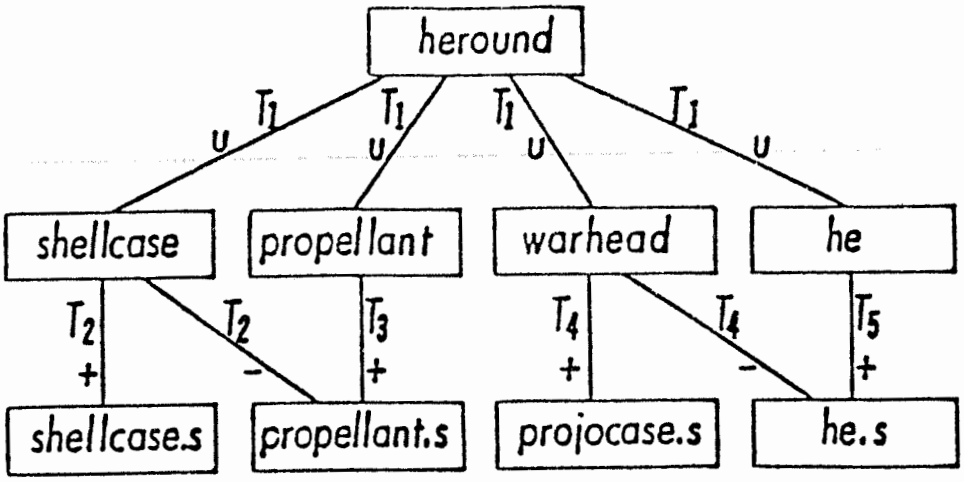


Figure 2. Sample Hierarchy of Object "heround"

Table II. Displaying Solids of Object "heround"

SOLID	TRANSFORMATION	PATH
shellcase.s	T2 * T1	heround/shellcase/shellcase.s
propellant.s	T2 * T1	heround/shellcase/propellant.s
propellant.s	T3 * T1	heround/propellant/propellant.s
projocase.s	T4 * T1	heround/warhead/projocase.s
he.s	T4 * T1	heround/warhead/he.s
he.s	T5 * T1	heround/he/he.s

---

c oldsolid newsolid

This command is used to produce and display a copy of a solid. In this case, the solid "oldsolid" will be copied into a solid called "newsolid." A new identical solid record is added to the data file. The solid parameters are copied as they appear in the solid record.

Examples:
```
c arb8 hullbot.s
c arb8 turrettop.s
c tgc wheelrim.s
c tor tirel.s
```

---

n old new

This command is used to rename objects in the data file. In this case, the object "old" will be renamed "new." A note of caution: The name is changed only in the object record itself, not in any member records. Thus, if the object "old" appears as a member of any other object, the name will not be changed there.

Examples:
```
n test hull
n g00 air
n gl turret
```

Z

This command clears (Zaps) the screen of all displayed objects.

---

g group obj1 obj2 ..... objn

This command creates or appends to a combination record and is used to group objects together either for editing or displaying purposes. In this case, "obj1" through "objn" are added as members to the combination "group." If "group" does not exist, it is created and "obj1" through "objn" are added as members.

Examples:
```
g shell hull turret
g tank wheels engine crew shell
g tank track
```

---

r region op1 sol1 op2 sol2 .... opn soln

This command is used to create regions or append to regions. If "region" exists, then solids "sol1" through "soln" are added as members with "op1" through "opn" as the defining operations. If "region" does not exist, then it is created and solids "sol1" through "soln" are added as members with "op1" through "opn" as the defining operations. A region is merely a combination record with a flag set and is distinguished from other combinations (groups) since it has meaning to the COMGEOM solid modeling system.

When a region is created, the item and air codes are set equal to zero. To change the item and air codes use the "I" command. Note: In the past all members of a region had to be solids, but recently regions have been allowed as members of regions. Hence, the names "soli" can also be regions now.

Examples:
```
r hulltop.r + hulltop.s - hullleft.s - hullright.s
r gun + gun.s - gunin.s
r gunair + gunin.s
```

---

i object combname brname

This command is used to make an instance of an object. An instance of an object is produced by creating a combination and making the object a member. In this case, an instance of "object" is made by creating the combination record "combname" (if "combname" does not already exist) and adding "object" as a member. This member also has a second name "brname" added to the member record, which can be thought of as the name of this branch in the hierarchical data structure. If "combname" already exists, then "object" is added as the next member and "brname" is added as the branch name.

An instance is used to refer to an object, without making actual copies of the object. Instances are useful when one is adding a certain component to a target description many times. Furthermore, any modifications to an object which has been instanced need only be done in the original (prototype) object. These modifications will then be automatically reflected in all the instances of the object.

Examples:
```
i heround ammo hel
i heround ammo he2
i heat ammo heatl
i heat ammo heat2
```

---

f face distance

This command allows the user to project a face of an arb (arbitrary polyhedron) being edited a normal distance to create a new arb. The value of "face" is 4 digits such as 1256. If the face is projected in the wrong direction use a negative "distance." One use for this command is for producing armor plates of a desired thickness.

Examples:
```
f 1234 20
f 2367 34.75
f 2367 -34.75
```

---

l object

This command is used to list information about objects in the data file. The information listed depends on what type of record "object" is. If "object" is a combination record, then the members are listed. If "object" is a solid record, then the GED general solid type and the parameters as presently in the data file are listed. Note: Only the solid parameters as they exist in the solid record are listed; no transformation matrix is applied. Hence, if the solid was edited as a member of a combination, the "l" command will not reflect the editing in the listed parameters.

```
Examples:      l hull
               l turret
               l turtop.s
               l arb8
```

---

m oldsolid newsolid axis

This command is used to create and display a new solid record which is the mirror image of an existing solid. The mirror image is about an axis and is created by changing the signs on the solid's parameters, depending on which axis the solid is mirrored about. In this case, a mirror image of the solid "oldsolid" will be created about the axis indicated by "axis," and the new solid record will be called "newsolid." The only acceptable values for the parameter "axis" are "x," "y," and "z."

```
Examples:      m tur.left.s tur.right.s y
               m tur.top.s tur.bot.s z
               m tur.front.s tur.back.s x
```

---

p dx [dy] [dz]

This command allows a user in the SOLID EDITING mode to input exact parameter modifications by the keyboard. The meaning of the values typed after the p command varies, depending on what solid editing feature is being used. Examples of this command will be given in the discussion on solid editing.

D comb mem1 mem2 .... memn

This command allows one to delete members from a combination record. In this case, members "mem1" through "memn" will be deleted from the combination "comb."

Examples:
```
D tank hull wheels
D region1 solid8 solid112
D turtop.r tursidel.s tursider.s turback.s
```

I region item air

This command allows one to change the item or air code numbers of a region. If the air code ("air") is not included, a zero is assumed. To change the air code, a zero item code should be used. (See second example below.)

Examples:
```
I region1 105
I region7 0 2
I region11 129 0
```

---

e object

This command is used to display (draw) objects on the screen. In this case, "object" will be displayed on the screen. Note that "object" must be found in the table of contents.

---

d object

This command is used to drop (delete) objects from the display. In this case, "object" will be deleted from the screen display. This command is opposite of the "e" command.

---

k object

This command is used to remove (kill) objects from the data file. In this case, "object" will be removed from the data file. Note the distinction between the "d" command which deletes objects from the display and the "k" command which removes objects from the data file.

---

t

This command produces a list of the table of contents of all objects in the data file. The objects are listed a screenfull at a time. A carriage return will produce the next screenfull.

---

a arbname rot fb

This command is used to create and display a new arb8 solid record. This new arb8 has two square (5 units x 5 units) parallel faces which are 0.5 units apart. These parallel faces are in planes whose orientations are defined by the two input angles - the rotation (rot) angle and the fallback (fb) angle. The orientation of armor plates are frequently defined using rotation and fallback angles. The vertex of this new arb8 is located in the center of the screen.

Examples:
```
a hullsec3.s 45 30
a tursec2.s 90 20
a topglacis.s 0 60
```

---

x

This command is the display debug command. It will produce a list of all objects that have been displayed, the paths traversed to draw the objects, and whether or not the objects are displayed in the present view.

---

q

This command is used to quit the graphics editor code. This is the normal halt.

Table III. Summary of GED Keyboard Commands

TASK	COMMAND
create/modify a region	r region opl soll .... opn soln
create instance of object	i object combinationname instancename
make copy of a solid	c oldname newname
rename object	n oldname newname
group objects	g groupname obj1 obj2 ..... objn
display (view) object	e object
list object information	l object
delete object from display	d object
remove object from desc.	k object
table of contents	t
zap (clear) the display	Z
debug-objects displayed?	x
solid parameter mods	p dx [dy dz]
mirror image of a solid	m oldsolid newsolid axis
project an arb face	f face distance
change region item/air codes	I region item air
delete members from comb.	D comb mem1 mem2 .... memn
define arb by rot,fb angles	a arbname rot fb
quit	q

VIII. PERIPHERAL DEVICES

Before we discuss the features of GED, we will introduce the hardware devices used to implement them. These devices are the "tools of the trade" for the GED user. We will discuss only basic operational characteristics here. Specific use of these devices will be covered in the later sections on the viewing and editing features of GED.

The JOYSTICK is a mechanical device used to do the rotations in GED. Any movement of the stick left or right rotates the display about the x-axis. Any movement of the stick up or down rotates the display about the y-axis. When the joystick is twisted in a clockwise or counterclockwise direction, the display rotates about the z-axis. Any combination motion of the stick will produce a "combined" rotation about the appropriate axes. All of these motions have a spring return to a null center position.

The FUNCTION SWITCH box contains thirty-two buttons. When any of the buttons is pressed, an action occurs or condition is set. Figure 3 depicts the functions programmed for each button. The buttons in the shaded area are used for editing while the rest are used for viewing the display. The exact functions assigned to these buttons will be discussed in the sections on viewing the display and on editing.

The CONTROL DIALS (knobs) are used to send digital information to the computer. As a knob is turned, a succession of numbers is available for use by the computer. Figure 4 depicts the functions assigned to each of the ten knobs. The exact functions of each of these knobs will be discussed in the angle distance cursor section and in the viewing features section.

The DATA TABLET is a graphics input device containing an X-Y coordinate grid which corresponds to the grid on the screen. Information is entered using a pen-like stylus. The distance this pen is from the tablet is important. If the pen tip is within one half inch of the tablet surface, the cursor location on the screen corresponds to the X,Y location of the pen on the tablet. This condition is called the "near" position. If the pen is more than one half inch from the tablet surface, the cursor remains located in the center of the screen. When the pen is pressed against the tablet surface, the pressure switch is activated and a bit is set which is a signal used by GED. The exact meaning of the pen near and pen press depends on what is being done and will be covered in the appropriate sections of this report.

IX. ANGLE DISTANCE CURSOR (ADC)

The angle distance cursor is a construction aid used to measure angles and distances. It should be noted that all measurements are made in the projected space of the screen, so one should measure only in a view normal to the surface where the measurement is to take place. The ADC is placed on (or removed from) the display by pushing the "ADC" function switch button. (See Figure 3.) The ADC consists of three cursors which cover the entire screen. Figure 5 depicts the ADC as it

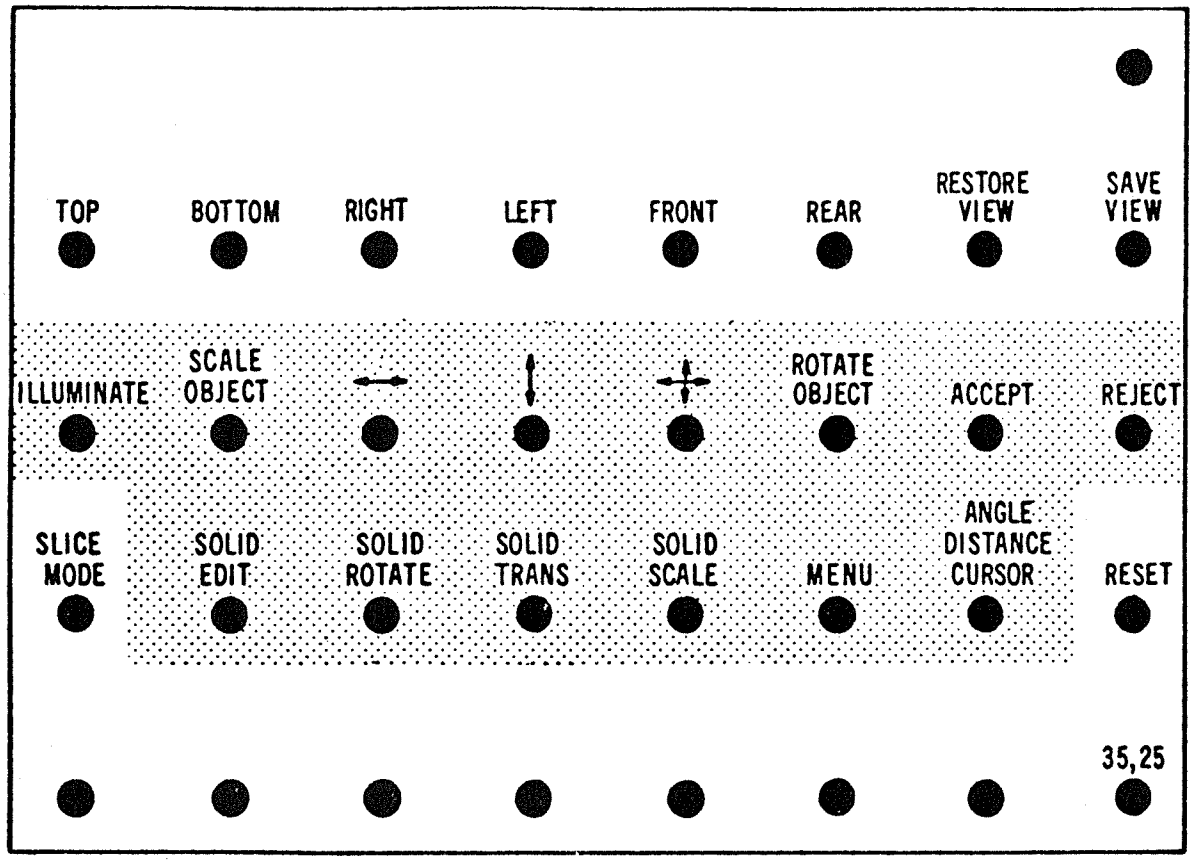


Figure 3. Tasks Assigned to the Function Switch Buttons

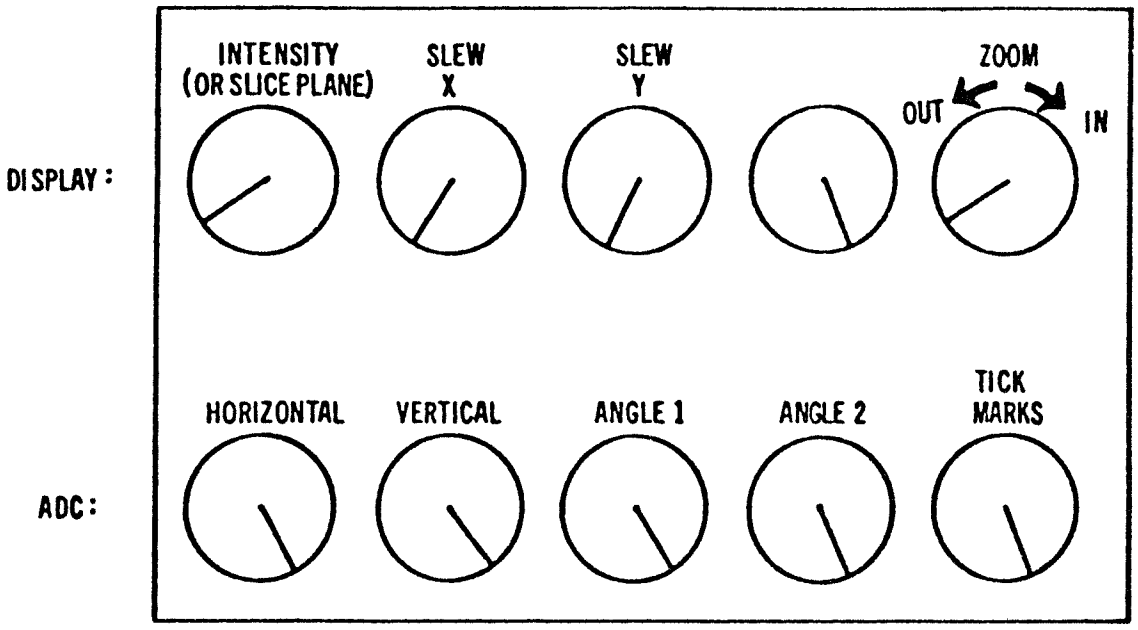


Figure 4. Functions Assigned to the Control Knobs

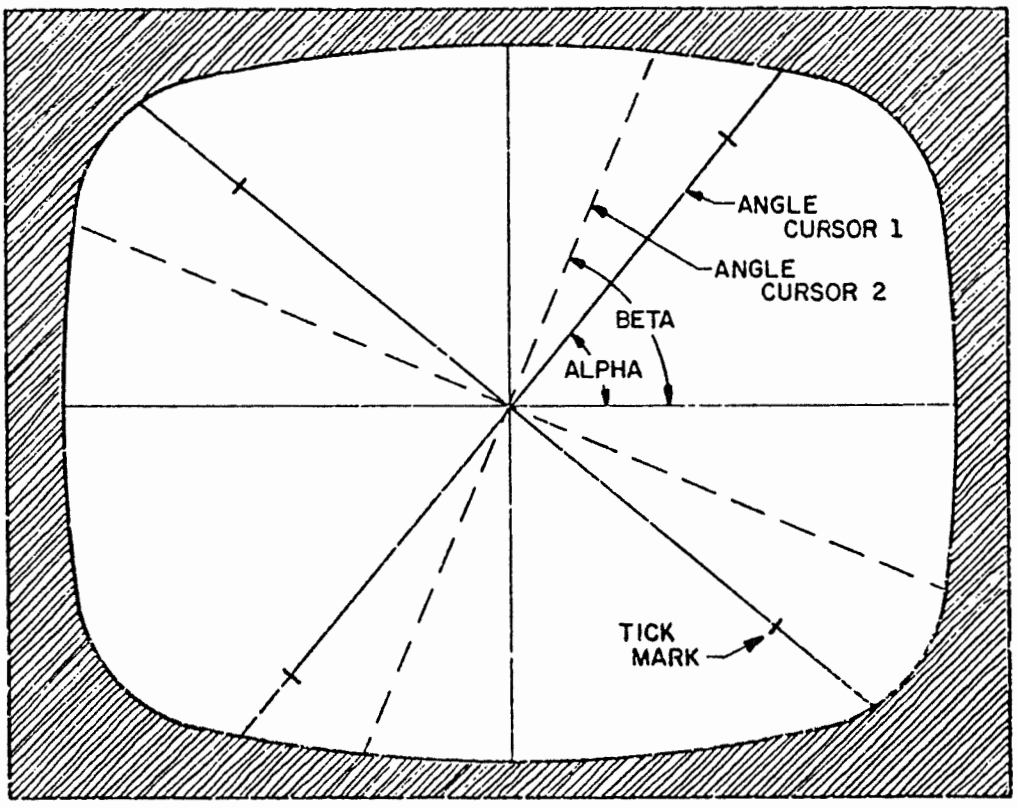


Figure 5. The Angle Distance Cursor

appears on the screen. All the cursors are centered at the same point and can be moved to any location on the screen. Two of these cursors rotate for angle measuring purposes. Angle cursor 1 is solid while angle cursor 2 is dashed. Angle cursor 1 has movable tic marks for measuring distances on the screen. The two angle cursors move with the horizontal and vertical lines of the main cursor. The resulting effect is the moving of the center point horizontally or vertically. The ADC is controlled by the bottom row of the control dial knobs (Figure 4):

Knob 6	moves the center in the horizontal direction
Knob 7	moves the center in the vertical direction
Knob 8	rotates angle cursor 1 (alpha)
Knob 9	rotates angle cursor 2 (beta)
Knob 10	moves the tic marks

Whenever the ADC is on the screen, there is a readout at the bottom of the screen listing pertinent information about the ADC. This information includes the angles that angle cursors 1 and 2 have been rotated (alpha and beta), the distance the tic marks are from the center of the ADC, and the location of the center of the ADC. This information is continually updated on the screen.

X. VIEWING FUNCTIONS

The GED viewing features are designed to allow one to examine a target description in close detail. Any of the viewing features can be invoked at any time. It should be noted that these functions do not change the actual data, only the way these data are displayed.

Six standard views (front, rear, top, bottom, left, and right) and one oblique view (azimuth 35, elevation 25) are each assigned to the function buttons. (See Figure 3.) Hence, any of these views is immediately available at the press of the appropriate function button. The views available are not limited to these standard views, however, as the display can be rotated to any view by using the joystick. By pressing the function button labeled "save view" (Figure 3), the present viewing aspect angle of the display is saved. At any time, the saved view can be immediately returned to the screen by pressing the "restore view" function button. (See Figure 3.) The "restore view" button will be lit whenever a view has been saved. The function button labeled "reset" (Figure 3) restores the display to the default view (front) when pressed.

The display can be panned or slewed on the screen in two ways - using the data tablet and pen or by using the control knobs. When one is editing, the tablet and pen are not available for slewing; hence, one must use the control knobs to slew the display. If one is NOT editing, then whenever the pen is pressed, the display moves in the direction of an imaginary vector drawn from the center of the screen to the cursor location on the screen. Recall that there is a one-to-one correspondence between the pen location and the cursor location. The longer this vector, the faster the display will move. To slew the display using the control knobs, one uses the knobs labeled "slew x" or

"slew y." (See Figure 4.) The null positions on these knobs is in the center or straight up. If the "slew x" knob is turned clockwise of center, the display will move to the right. If it is turned counterclockwise, the display will move to the left. For the "slew y" control knob, clockwise of the center moves the display up and counterclockwise moves the display down. The further these knobs are turned from center, the faster the display moves.

One can zoom the display by using the control knob labeled "zoom." (See Figure 4.) Again, the null position of this knob is center or straight up. Turning this knob clockwise of center causes the display to increase in size, producing a zoom-in effect. Turning this knob counterclockwise of center causes the display to decrease in size or zoom out. Again, the further the "zoom" knob is turned from center, the faster the zooming will occur.

The viewing features of GED also include a slice mode. To invoke the slice feature, one presses the "slice mode" function button. (See Figure 3.) An imaginary slicing plane, parallel to the screen, can then be moved about by turning the first control knob. (See Figure 4.) Turning this knob clockwise moves the imaginary plane into the screen, and counterclockwise moves the plane out of the screen. All portions of the display in front of the plane are eliminated from the display. The result of moving this plane into the screen is a slicing effect with the parts of the display closest disappearing first. To leave the slice mode, just press the "slice mode" button again.

XI. SELECTING OBJECTS FOR EDITING

Before we discuss the editing features of GED, we will discuss how one selects objects for editing. To be edited, an object must be displayed on the screen. Since the object to be edited may be only a small portion of the display, a procedure is needed to select an object from the many objects being displayed. This procedure relies on the hierarchical structure of the GED data base. As stated before, when objects are displayed, every path of every object is traversed and the bottom object, which is always a solid, is drawn. Hence, one can think of each solid drawn not as one solid, but as the unique path used to reach that solid. As an example, the six paths (solids) for the sample object "heround" were presented in Table II.

To select an object for editing, one first must enter the illuminate mode by pressing the function button labeled "illuminate." (See Figure 3.) The data tablet and pen are then used to select the object. The surface of the data tablet is divided into imaginary horizontal strips. Each of these strips corresponds to a path (solid) drawn on the screen. For example, if the sample "heround" were displayed, the tablet surface would be divided into six horizontal strips. The pen is used in the "near" mode (within one half inch of tablet surface) to illuminate the paths. The path corresponding to the pen location is written in the upper right-hand corner of the screen. The solid at the bottom of this path is illuminated (drawn brighter) on the screen. In this manner one can "scan" the complete display by

simply moving the pen up and down the tablet in the near mode. Hence, the illuminate mode is also quite useful just to identify objects in a description. However, the primary purpose of the illumination mode is to select objects for editing.

The actual selecting of an object for editing is a two-step process. First, one selects the path desired, then one selects the particular object of interest from that path. One "selects" by pressing the pen on the tablet surface. When a path is selected, the word PATH followed by the members of that path is written in the upper right of the screen. At that point, one is ready to select a specific member (object) from that path for editing. Again, the tablet is used for this selection. The tablet surface is now divided into as many rows (zones) as there are members in the selected path. When the pen, in the near mode, is in a path member's assigned zone on the tablet, that member's name will be illuminated in the path listing. When the specific member to be edited is illuminated, it is selected by pressing the pen. The word EDIT will now appear in the upper right of the screen ahead of the path listing. One is now ready to edit this selected object.

The only way out of the EDIT mode is to accept or reject any editing performed. This is done by pressing the function buttons labeled "accept" or "reject." (See Figure 3.) If at any time in the selection process an incorrect object is selected, just press the "reject" button and begin the selection process again.

XII. OBJECT EDITING

The heart of the GED system is its editing features. The editing features are divided into two classes: object editing and solid editing. Object editing is designed to allow one to change the location, size, and orientation of an object. Recall that an object is defined as the basic data unit of the GED system and includes both combinations and solids. In the case of a solid, one needs to change not only its location, size, and orientation, but also its "shape." Changing the shape of a solid means changing any of its individual parameters. Hence, solid editing is handled separately.

Any object in the data file may be selected for object editing. The object is picked using the object selection procedure discussed earlier. Recall that in the GED data structure, all members of combinations have transformation matrices associated with them, allowing assemblies to be edited as units. Object editing is the vehicle for performing such tasks. All object editing is done using homogeneous transformation matrices. When the editing is accepted, new transformation matrices are created in all appropriate member locations. If the object happens to be a single-member path (solid), then the matrix is applied to the solid's parameters. Otherwise, all object editing is stored in transformation matrices in the data base.

Once an object is selected for editing, the second row of function buttons (Figure 3) is used to select the editing to be done. Then either the data tablet and pen or the joystick is used to perform the

actual editing.

To SCALE a selected object, one first presses the function button labeled "scale object." The data tablet and pen are then used to scale the object. The object is scaled about the target origin; thus, unless the object is centered at the origin, some translation of the object will occur. The location of the pen on the tablet when pressed determines the scale factor used. If the pen is above the center of the screen, the scale factor will be greater than one. The further the pen is located above the center, the larger the scale factor. Likewise, if the pen is located below the vertical center, the scale factor will be less than one and the further below the center, the smaller the scale factor. As long as the pen is pressed, the object will continue to increase or decrease in size.

To ROTATE a selected object, one first presses the function button labeled "rotate object." The joystick is then used to rotate the object. The object is rotated about the center of the view (screen). If one wishes to rotate the object about another point of interest, the entire display can be slewed until the desired point is positioned at the center of the screen.

To TRANSLATE a selected object, one has three function buttons from which to select. (See Figure 3.) The data tablet and pen are then used to translate the object. The center of the object "follows" the pen (cursor) location whenever the pen is pressed. The three function buttons define the direction allowed for movement of the object. The button labeled with a horizontal arrow (⟷) allows the object to follow only the horizontal location of the pen. The vertical location of the object does not change. The function button labeled with a vertical arrow (↕) allows the object to follow only the vertical location of the pen. The horizontal location of the object does not change. The function labeled with crossed arrows (⟷↕) allows the object to follow the X,Y location of the pen.

The object editing features can be invoked in any order and at any time once an object has been selected for editing. During object editing, any of the viewing features, such as changing views, zooming, and slewing, can be used and, in fact, are usually quite useful. Again, the only way to exit the object editing mode is to accept or reject the editing. If the "reject" button is pressed, the object will return to its pre-edit state. If the "accept" button is pressed, the data base will be changed to reflect the object editing performed.

XIII. SOLID EDITING

The solid editing feature allows the user to interactively translate, rotate, scale, and modify individual parameters of a solid. Whenever one is in the solid edit mode, the parameters of the solid being edited are listed and continually updated at the bottom of the screen. Certain parameters are also labeled on the solid being edited. Solid editing is generally used to "build" objects by producing solids of the desired shape and size in the correct orientation and position.

Once the object is built, object editing is used to scale, orient, and position the object in the description. The general philosophy of solid editing is to first create a copy of a prototype solid with the desired name and then to edit this solid. The prototype solids should not be edited. As an example, suppose one were to build the sample object "heround" mentioned earlier. To produce the base of the shell, one would need a cylindrical shaped solid. The prototype solid is the TGC (Table 1), so one would type:

```
c tgc shellcase.s
```

A new solid record called shellcase.s would be created and displayed on the screen. This shellcase.s solid would then be edited using solid editing to produce the exact solid parameters desired. The solid record tgc would be unchanged and available for copying the next time a cylindrical solid is needed.

The procedure for solid editing is quite similar to that for object editing. First, the solid must be selected for editing. A solid is selected for editing using the illuminate mode, just as in object editing, except a solid must be selected. Recall that the bottom object in every hierarchy path is a solid. Second, one must push the function button labeled "solid edit." (See Figure 3.) This button sets up the solid edit mode: the solid parameters are listed at the bottom of the screen, pertinent solid parameters are labeled on the display, the solid editing function buttons are activated, and a menu header depending on the solid type is written on the right side of the screen. The meaning of the menu will be discussed shortly. Third, one selects (by pressing the appropriate function buttons) and performs the editing desired. Finally, one exits the solid editing mode by accepting or rejecting the editing performed, just as in object editing.

Recall, in the section on object editing it was mentioned that since a solid is indeed an object, a solid can be edited (translated, rotated, and scaled) as an object. Hence, the solid editing mode is really only necessary to perform parameter modifications. However, much of the object editing is done about the target or screen center, which can be annoying when editing a solid. Also, precise parameter modifications are possible (using the "p" keyboard command) in solid editing. Thus, it was deemed desirable to include translation, rotation, and scaling in solid editing also.

Solid TRANSLATION allows the user to place the solid being edited anywhere in the description. To invoke this option, one presses the function button labeled "solid trans." (See Figure 3.) To move the solid, use the data tablet and pen. Whenever the pen is pressed, the VERTEX of the solid moves to that location on the screen. One can read the actual coordinates of the vertex on the bottom of the screen along with the other parameters. If the actual desired coordinates of the vertex are known, one can place the solid exactly using the "p" keyboard command. For example, to place a solid's vertex at the coordinates (x, y, z) one would type

```
p x y z
```

The solid would then jump to this location.

The solid SCALE feature allows the user to scale the solid being edited to any desirable size. The scaling is done about the vertex of the solid; hence, NO translation of the solid occurs. The scaling is performed using the data tablet and pen just as in object scaling. One can input an exact scale factor using the "p" keyboard command. For example, typing

p factor

will scale the solid by an amount equal to "factor." The value of "factor" is absolute - the original solid is scaled. The size of the original solid is defined when the "solid scale" function button is pressed. By setting "factor" equal to one, the original size solid will be displayed on the screen.

Solid ROTATION allows the user to rotate the solid being edited to any desired orientation. The rotation is performed about the vertex of the solid. To select this option, one presses the function button labeled "solid rotate." (See Figure 3.) The actual rotation is done using the joystick. One can input exact angles to rotate the solid by using the "p" keyboard command. For example, typing

p alpha beta gamma

will rotate the solid "alpha" degrees about the x-axis, "beta" degrees about the y-axis, and "gamma" degrees about the z-axis. Alpha, beta, and gamma are measured from the original "zero" orientation of the solid, defined when the "solid edit" function button was pressed. Hence, typing

p 0 0 0

will always return the solid to its original position (its position when the current solid editing session began).

The PARAMETER editing feature allows the user to modify any of the parameters which comprise the selected solid. To invoke this feature, one presses the function button labeled "menu." (See Figure 3.) A menu (listing) of parameters to edit will then appear on the right side of the screen under the menu header. The contents of this menu, of course, depend on the type of solid being edited. To select an item (parameter) from this menu, one uses the data tablet and pen. The immediate region (strip) surrounding each item of the menu "belongs" to that item. Whenever the pen is in one of these regions, that menu item will get brighter in the menu listing on the screen. A menu item is picked by pressing the pen when that item name is illuminated. It should be noted that whenever the pen is in the "menu area," the code assumes the user is attempting to pick a menu item. Hence, the menu area of the screen (the far right edge) can NOT be used by the pen for editing while the menu is on the screen.

For parameter editing, the solid type determines the menu items. The GED code recognizes four general solid types for parameter editing. All the specific solid types which fall into one of these general classes are treated as the general type. For example, all the cylindrical solid types are treated as the general type - the TGC. In the following paragraphs, we will discuss parameter editing for each of these general types of solids.

A. ARB PARAMETER EDITING

The GENERAL ARB class of solids represents all the convex polyhedrons (RPP, BOX, RAW, and ARBs). The ARBs comprise five classes of polyhedrons each with a characteristic number of vertices. These are the ARB8, ARB7, ARB6, ARB5, and ARB4, where the ARB8 has eight vertices, etc. GED handles all these ARBs as ARB8s, but with some vertices equal. All the eight vertices of the GENERAL ARB are labeled on the screen, although in certain cases (RAW,ARB7,ARB6,ARB5,ARB4) some vertices will be equal. Figure 6 depicts typical ARBs as they would be displayed and labeled during solid editing.

To edit the ARB vertices, EDGES are moved. Hence, the GENERAL ARB menu is a list of edges to move:

```
GENERAL ARB 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
```

The edge to be moved is selected from this menu and is then moved using the data tablet and pen. When the pen is pressed, the line containing the selected edge is moved so it goes through the pen location and is parallel to the original edge. The endpoints of the new edge are then calculated to remain in the "bounding" planes of that edge. In the case of the ARB8, for example, the bounding planes for edge 58 are planes 4378 and 1265.(See Figure 6.)

The information printed on the bottom of the screen is the x, y, z coordinates of the eight vertices:

vertex	coordinates
1,2	x1 y1 z1 x2 y2 y3
3,4	x3 y3 z3 x4 y4 z4
5,6	x5 y5 z5 x6 y6 z6
7,8	x7 y7 z7 x8 y8 z8

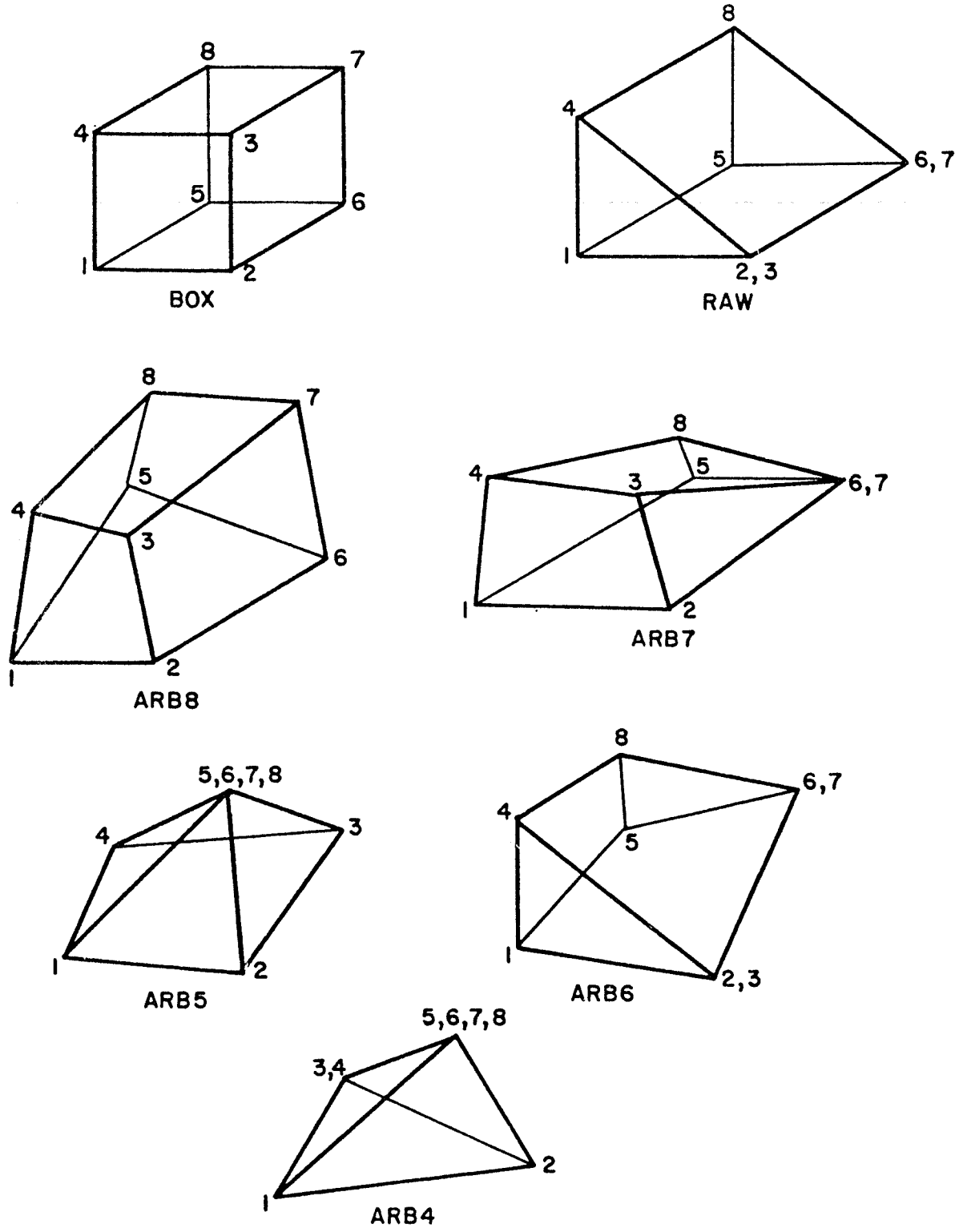


Figure 6. Typical Generalized ARBs

These values are continually updated during solid editing.

The user can force the line containing an edge through any point using the "p" keyboard command. Hence, typing

p x y z

will force the line containing the edge being edited to pass through the point (x, y, z). No check is made for concave ARBs or for ARBs with crossed edges. Visual feedback from the display is considered adequate to detect these situations.

B. TGC PARAMETER EDITING

The TGC general class of solids includes all the cylindrical COMGEOM solids. The defining parameters of the TGC are two base vectors (A and B), a height vector (H), two top vectors (C and D), and the vertex (V). Reference 1 contains more information on these parameters. The top vectors C and D are directed the same as the base vectors A and B, respectively; hence, the top vectors are defined only by their lengths (c and d). Figure 7 depicts these parameters on typical TGCs. When a TGC is edited in the solid editing mode, only vectors A and B are labeled on the display.

All the vectors (A,B,C,D,H) of the TGC can be scaled during parameter editing. In addition, the height vector (H) and the base (AxB) can be rotated. These parameter editing options are reflected in the TGC menu:

TGC MENU
scale H
scale A
scale B
scale c
scale d
rotate H
rotate AxB

The scaling of the lengths of the vectors is done using the data tablet and pen in the same manner as object editing. Exact vector lengths can be achieved easily using the "p" keyboard command. If one is scaling a vector, then typing

p x

will make the vector being scaled have a length equal to "x."

To rotate vector H or surface AxB, one uses the joystick. Since vectors C and D have the same direction as vectors A and B, respectively, the top and bottom surfaces of the TGC will remain parallel. Hence, rotating the base surface (AxB) will also rotate the top surface. The "p" keyboard command is not used with the parameter rotation of the TGC.

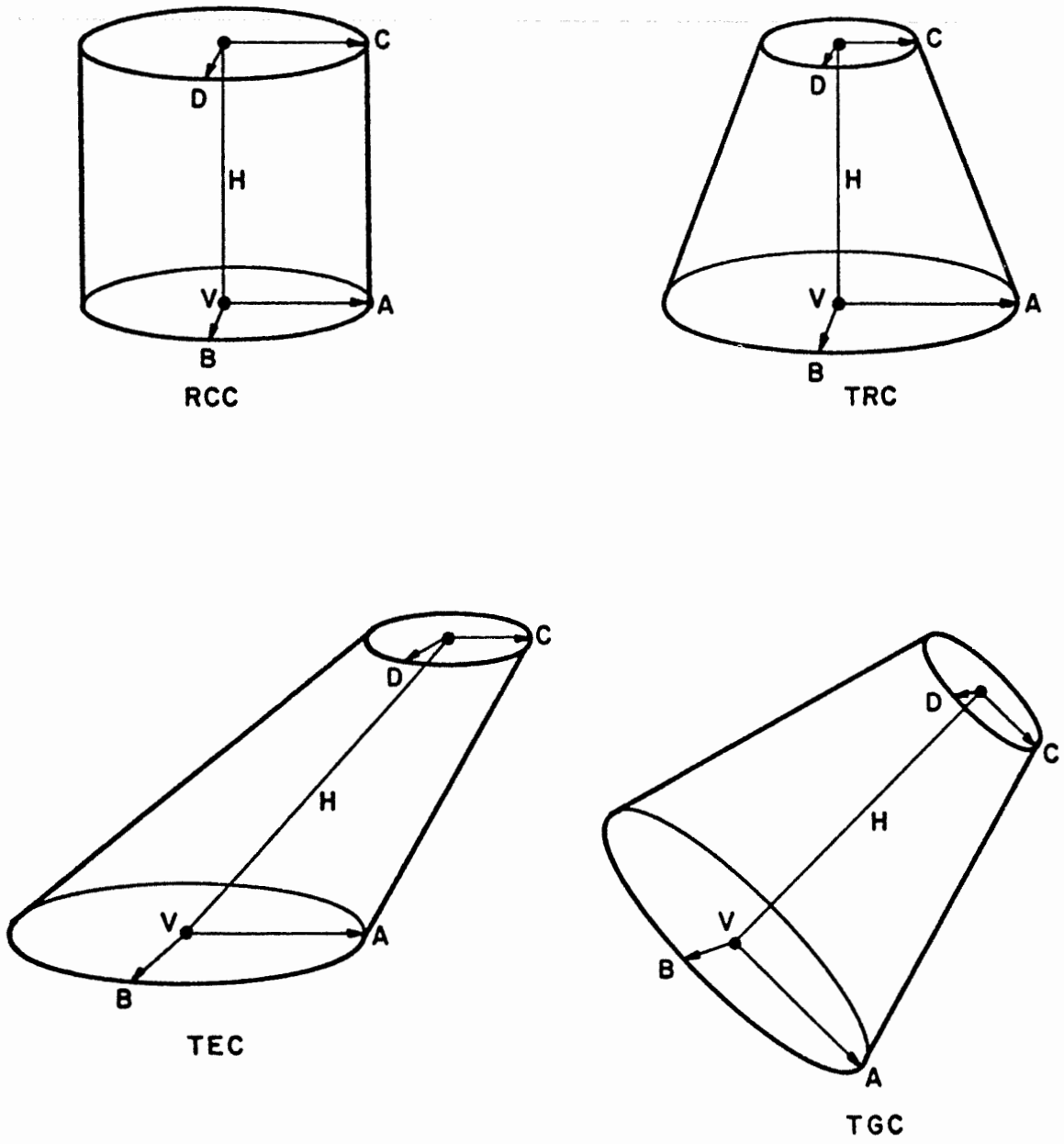


Figure 7. Typical TGCs

During solid editing, the parameters of the TGC are constantly updated on the bottom of the screen:

```
V    x y z
H    x y z |H| a b g rot fb
A    x y z |A|
B    x y z |B|
c    |c|
d    |d|
AxB a b g rot fb
```

where "a," "b," "g" are the direction cosines and "rot" and "fb" are the rotation and fallback angles of the listed vector.

C. ELLG PARAMETER EDITING

The ELLG general class represents all the ellipsoidal solids, including spheres and ellipsoids of revolution. The defining parameters of the ELLG are three mutually perpendicular vectors (A, B, and C) and the vertex (V). Reference 1 contains a more complete definition of these parameters. Figure 8 depicts typical ELLGs with these parameters labeled. When an ELLG is being edited, only vectors A and B are labeled on the display.

The parameter editing of the ELLG consists of scaling the lengths of the vectors A, B, C. These parameter editing options are listed in the menu:

```
ELLG MENU
 scale A
 scale B
 scale C
```

The scaling of these vectors is done using the data tablet and pen in exactly the same manner as in object scaling. The "p" keyboard command again can be used to produce a vector of desired length.

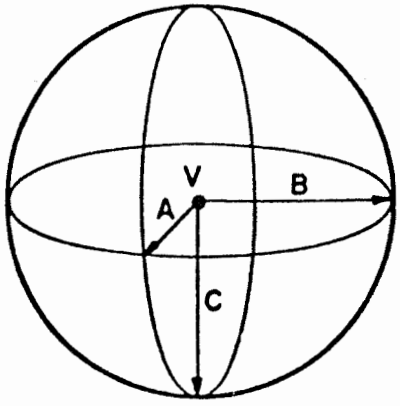
The parameters of the ELLG are continually updated on the bottom of the screen during solid editing:

```
V    x y z
A    x y z |A| a b g rot fb
B    x y z |B| a b g rot fb
C    x y z |C| a b g rot fb
```

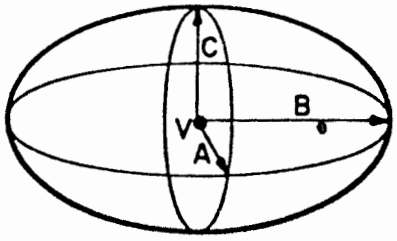
where "a," "b," "g" are the direction cosines and "rot" and "fb" are the rotation and fallback angles of the listed vector.

D. TOR PARAMETER EDITING

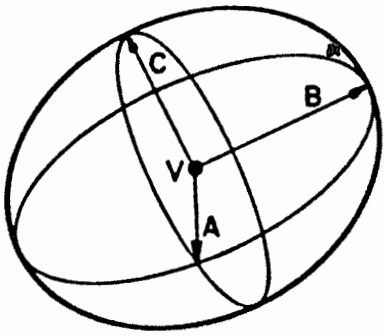
The TOR general class of solids contains only one type of torus, one with circular cross sections. The defining parameters of the TOR



SPH



ELLG



ELLG

Figure 8. Typical ELLGs

are two radii (r1 and r2), a normal vector (N), and the vertex (V). The scalar r1 is the distance from the vertex to the midpoint of the circular cross section. The scalar r2 is the radius of the circular cross section. The vector N is used to orient the torus. Reference 1 contains a more detailed definition of these parameters. Figure 9 depicts these parameters. During solid editing, none of these parameters are labeled on the screen.

The parameter editing of the TOR consists of scaling the radii; hence,the menu contains only two members:

TORUS MENU
scale radius 1
scale radius 2

As before, the scaling of these radii are done using the data tablet and pen and the "p" keyboard command can be used to produce a radius of any desired length.

The parameters of the TOR are continually updated on the bottom of the screen during solid editing:

V x y z
r1 |r1|
r2 |r2|
N x y z a b g rot fb

where "a", "b", "g" are the direction cosines and "rot" and "fb" are the rotation and fallback angles of the vector N.

Table IV presents a summary of parameter editing for each of the general solid classes.

XIV. CONCLUSION

BRL has developed the Army's first interactive solid modeling system, called GED. GED performs two basic functions: viewing and editing. The standard viewing capabilities of zooming, slewing, slicing, and rotation are available. Likewise, all the standard editing features are also available. The user easily traverses the hierarchical data structure, applying the editing functions of rotation, translation, and scaling to any position in the hierarchy. The hierarchical structure can be modified and regrouped and regions created and modified. Specific parameter editing can also be applied to the solids to produce any shape solid desired. GED has made great gains in reducing the bottleneck the target description process has been creating in the vulnerability process at BRL.

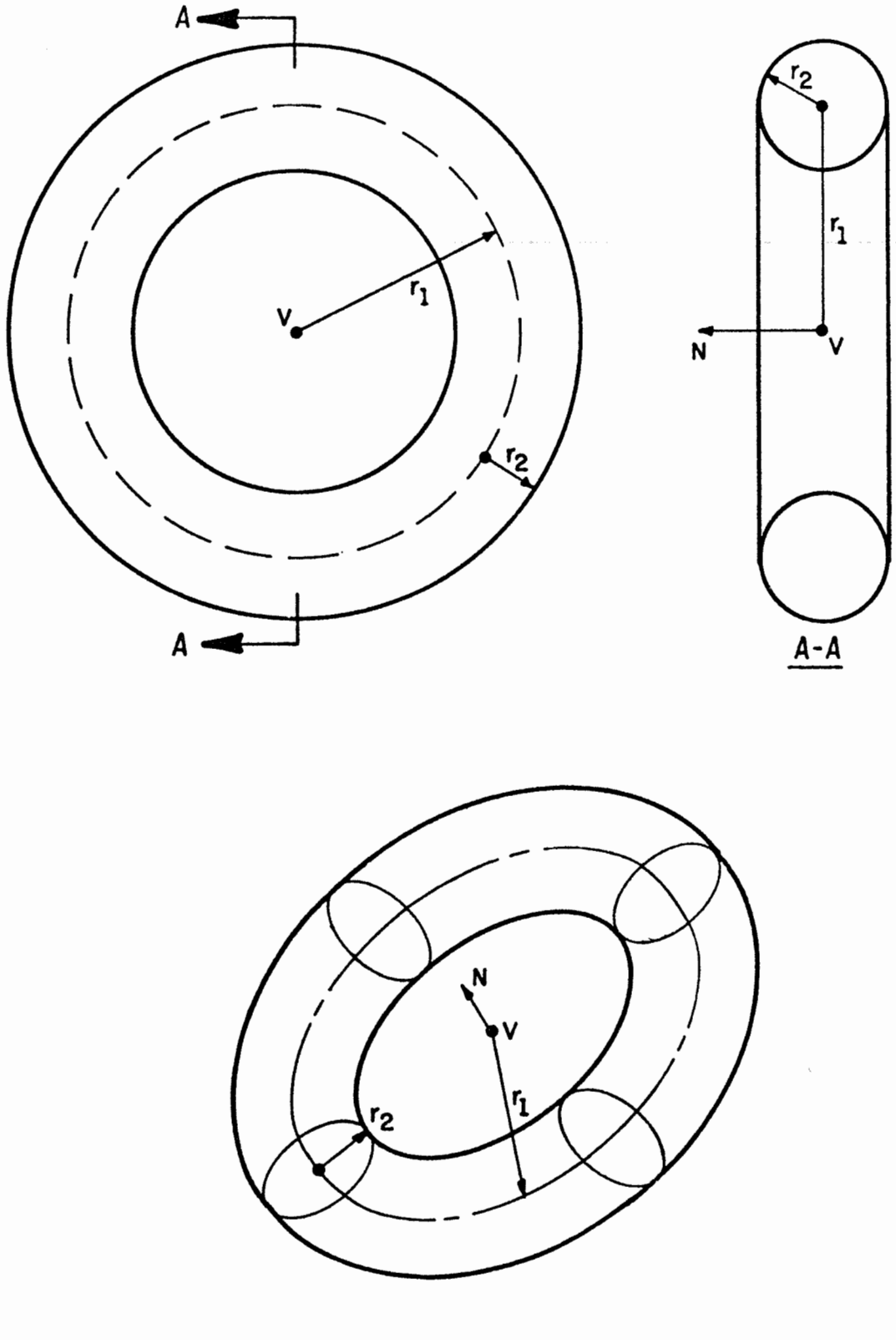
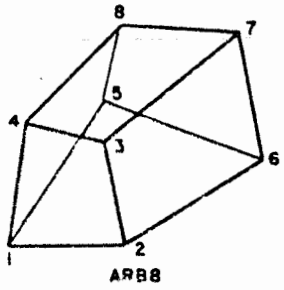
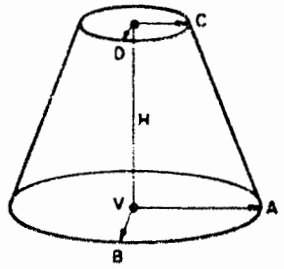
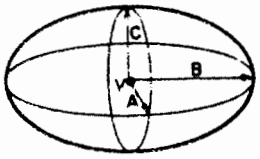
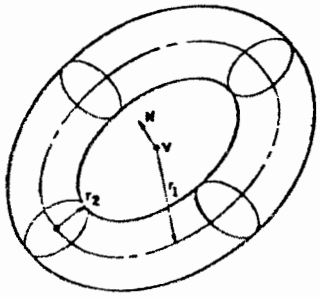


Figure 9. TOR Parameters

TABLE IV. Parameter Editing Summary

SOLID	MENU	PARAMETER INFO PRINTED
 ARB8	GENERAL ARB 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	1,2 x1 y1 z1 x2 y2 z2 3,4 x3 y3 z3 x4 y4 z4 5,6 x5 y5 z5 x6 y6 z6 7,8 x7 y7 z7 x8 y8 z8
 TGC	TGC MENU scale H scale A scale B scale c scale d rotate H rotate AxB	V x y z H x y z |H| a b g rot fb A x y z |A| B x y z |B| c |c| d |d| AxB a b g rot fb
 ELLG	ELLG MENU scale A scale B scale C	V x y z A x y z |A| a b g rot fb B x y z |B| a b g rot fb C x y z |C| a b g rot fb
	TORUS MENU scale radius 1 scale radius 2	V x y z r1 |r1| r2 |r2| N x y z a b g rot fb

5.

MGED: Multi-device Graphics Editor

MGED

M ulti-device
G raphics
ED itor

VITAL STATISTICS	
CPUs	VAX and GOULD (unix)
DISPLAYs	Vector General 3000 Megatek 7250 Raster Tech One/80 Evans & Sutherland PS300 Megatek Merlin 9200 Tektronix 4014 Tektronix 4109 Silicon Graphics IRIS Family Any UNIX-plot Filter

DEFINITIONS

OBJECT -- Basic MGED data unit.
SOLID -- Primitive (comgeom).
COMBINATION -- Group of objects.
REGION -- Special combination (comgeom).

OBJECTS

ALL objects are referenced by NAME. Each name must be unique and be less than 16 characters. When MGED is invoked, an in-core directory is compiled which contains the name and disk address of every object in the file. When access to an object is necessary, the object's name is located in the directory and the object's disk address is used to locate the object.

The MGED data file is a BINARY file of sequential 128 byte blocks (records) of data. The actual organization of the data in each record depends on the type of record.

Types of records:

IDENT
SOLID
ARS Header
ARS Data
COMBINATION
MEMBER
POLYGON Header
POLYGON Data
B-SPLINE Header
B-SPLINE Knot Vector
B-SPLINE Control Mesh
MATERIAL
FREE

IDENT Record

The IDENT record is the first record in the file.

IDENT Record	
Variable	Meaning
i_id	'I'
i_units	current local units
i_version	version of this data
i_title[72]	title of this description

SOLID Record

The SOLID record contains the comgeom solid parameter data. All comgeom solids, except for the ARS are stored using solid records. The solid parameter data are stored as a vertex and vectors from that vertex.

SOLID Record	
Variable	Meaning
s_id	'S'
s_type	General solid type (genarb8, gentgc, etc)
s_name[16]	name of this solid
s_cgtype	COMGEOM solid type (rcc, sph, arb6, etc)
s_values[24]	solid parameters (vertex and up to 7 vectors)

SOLID RECORD EXAMPLES

sample.rcc V = (1, 2, 3) H = [0, 0, 8] r = 2.5

sample.ell1 V = (0, 0, 0) A = [10, 0, 0] r = 3

sample.rpp -10 10 -11 11 -12 12

```
S 18 sample.rcc 12
   1.0000    2.0000    3.0000    0.0000    0.0000    8.0000
   2.2097   -1.1692    0.0000    1.1692    2.2097    0.0000
   2.2097   -1.1692    0.0000    1.1692    2.2097    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

S 19 sample.ell1 10
   0.0000    0.0000    0.0000   10.0000    0.0000    0.0000
   0.0000    2.1213   -2.1213    0.0000    2.1213    2.1213
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

S 20 sample.rpp 1
  10.0000  -11.0000  -12.0000    0.0000   22.0000    0.0000
   0.0000   22.0000   24.0000    0.0000    0.0000   24.0000
 -20.0000    0.0000    0.0000  -20.0000   22.0000    0.0000
 -20.0000   22.0000   24.0000  -20.0000    0.0000   24.0000
```

ARS Header Record

The ARS primitive contains more than 8 vectors of information, hence the solid record cannot be used. Instead 2 record types are used to store the ARS. The first is the ARS header record and the second is the ARS data record.

ARS Header Record	
Variable	Meaning
a_id	'A'
a_type	ARS
a_name[16]	name of this ARS
a_m	number of curves
a_n	number of points/curve
a_curlen	number of granules (8 vectors) per curve
a_totlen	total number of granules for this ARS

Immediately following the ARS header record are "a_totlen" ARS data records. Just as with other solids, the ARS is stored as a vertex and vectors from that vertex.

ARS Data Record

This record contains up to 8 vectors of data to define an ARS.

ARS Data Record	
Variable	Meaning
b_id	'B'
b_type	ARSCONT
b_n	current curve number
b_ngranule	current granule for this curve
b_values[24]	up to 8 vectors of data

ARS RECORDS
EXAMPLE

```
A 21 sample.ars 3 9 2 6
B 22 1 1
   0.0000    0.0000   -1.5000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
B 22 1 2
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
B 22 2 1
  10.0000    0.0000    1.5000    2.0000    2.0000    1.5000
   0.0000   10.0000    1.5000   -2.0000    2.0000    1.5000
 -10.0000    0.0000    1.5000   -2.0000   -2.0000    1.5000
   0.0000  -10.0000    1.5000    2.0000   -2.0000    1.5000
B 22 2 2
  10.0000    0.0000    1.5000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
B 22 3 1
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
B 22 3 2
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
   0.0000    0.0000    3.0000    0.0000    0.0000    3.0000
```

COMBINATION Record

The COMBINATION record is used to store all "non-solid" objects. It is really a header record followed immediately by the "defining" member records.

COMBINATION Record	
Variable	Meaning
c_id	'C'
c_flags	'R' if is a region, blank otherwise
c_name[16]	name of this combination
c_regionid	region ident number
c_aircode	region air code number
c_length	number of members in this combination
c_material	region material code number
c_los	region los %

There are "c_length" member records immediately following each combination record.

MEMBER Record

There is one MEMBER record for each member of a combination record. The member records are stored sequentially, directly following "their" combination record.

MEMBER Record	
Variable	Meaning
m_id	'M'
m_relation	boolean operation for this member (u, +, -)
m_instname[16]	name of this member (must be in "directory")
m_mat[16]	transformation matrix for this member

COMBINATION RECORD EXAMPLES

```
region1 = + sample.rcc - sample.rpp

region2 = + sample.ell1

group = region1 and region2


C R region1 1000 0 2 1 100
M + sample.rcc
1.000000  0.000000  0.000000  0.000000
0.000000  1.000000  0.000000  0.000000
0.000000  0.000000  1.000000  0.000000
0.000000  0.000000  0.000000  1.000000
M - sample.rpp
1.000000  0.000000  0.000000  0.000000
0.000000  1.000000  0.000000  0.000000
0.000000  0.000000  1.000000  0.000000
0.000000  0.000000  0.000000  1.000000

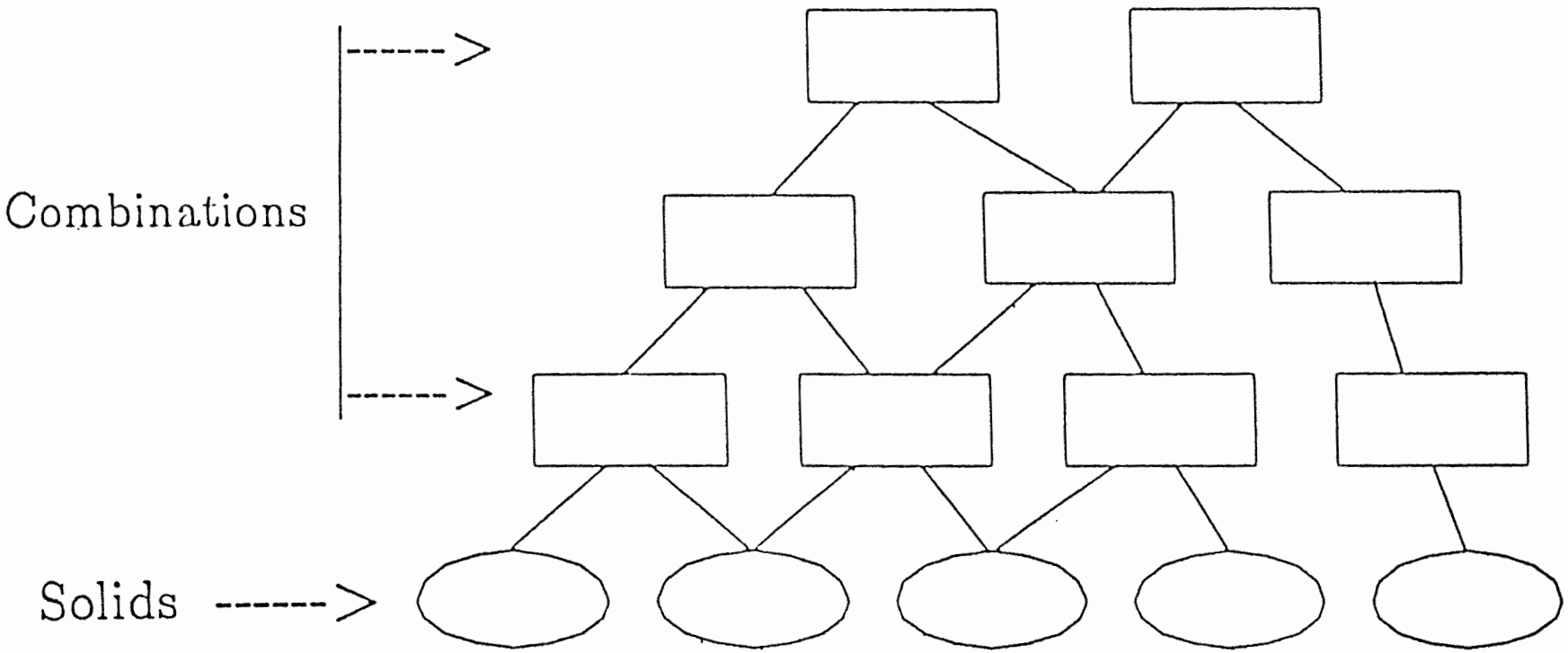
C R region2 1001 0 1 1 100
M + sample.ell1
1.000000  0.000000  0.000000  0.000000
0.000000  1.000000  0.000000  0.000000
0.000000  0.000000  1.000000  0.000000
0.000000  0.000000  0.000000  1.000000

C  group -1 0 2 0 0
M u region1
1.000000  0.000000  0.000000  0.000000
0.000000  1.000000  0.000000  0.000000
0.000000  0.000000  1.000000  0.000000
0.000000  0.000000  0.000000  1.000000
M u region2
1.000000  0.000000  0.000000  0.000000
0.000000  1.000000  0.000000  0.000000
0.000000  0.000000  1.000000  0.000000
0.000000  0.000000  0.000000  1.000000
```

MGED HIERARCHY

The MGED file structure is hierarchical. Each object is a member of another object or by definition it is a top level object. Objects are generally grouped together by function, hence objects are created which represent subsystems of the target. These subsystems are then grouped into higher level objects until one object represents the complete target system. The user has the freedom to name objects as he wishes, however certain conventions have been established. These conventions are concerned only with the top few levels of the hierarchy.

MGED HIERARCHY



TOP Level Naming Conventions (Armor)

The top level names are based on what the target is intended to be used for. Each MGED file should contain at least one of these names. The hierarchy for each top level name will employ like lower level objects. Using this scheme, several descriptions of a target can coexist, sharing like components.

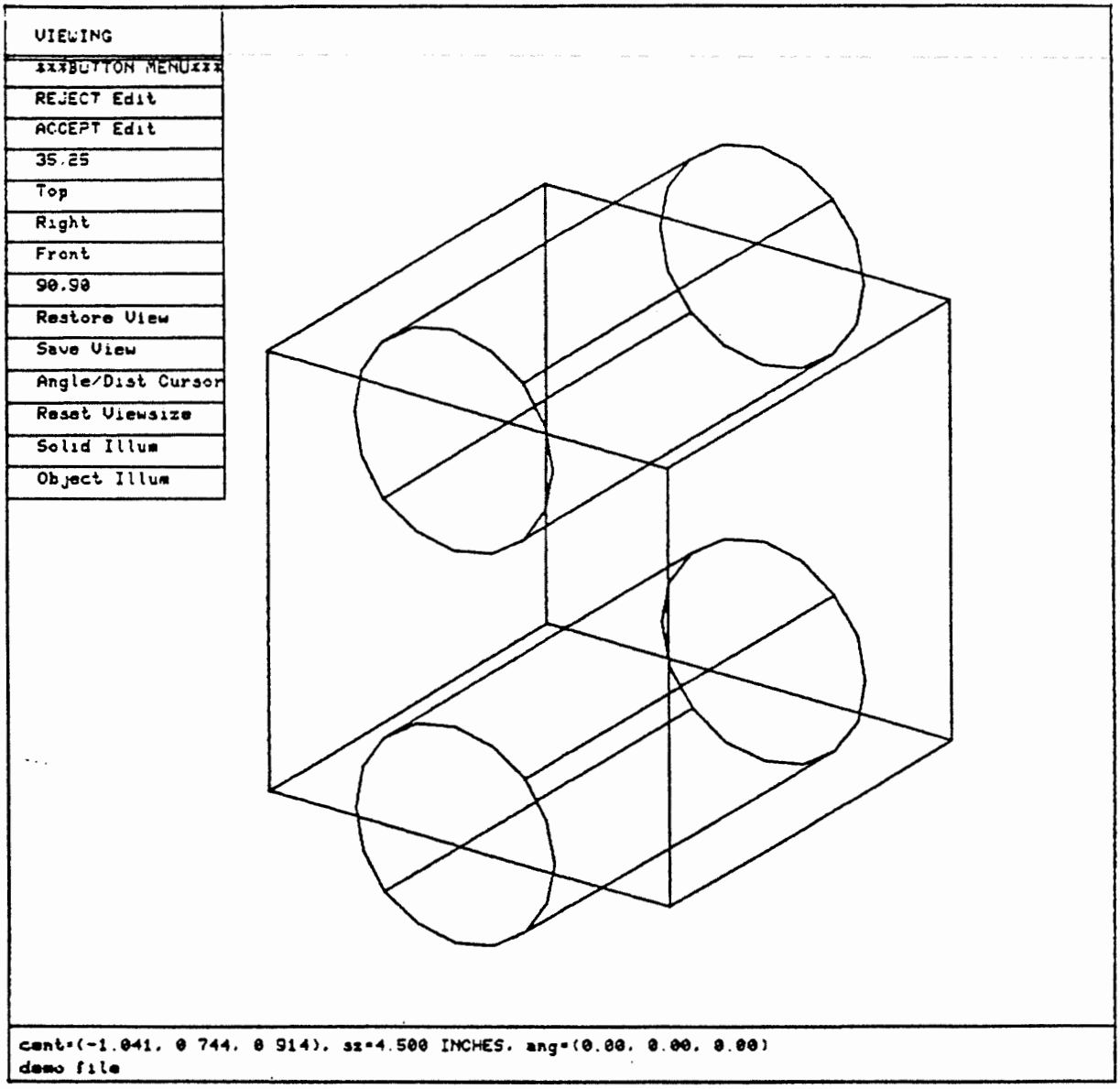
TOP Level Names
nuclear (nu)
compartment (ca)
component (co)
sparc (sp)
infrared (ir)
radar (ra)

The 2 letters in parentheses following each top level name are to be used to distinguish lower level objects.

LOWER Level Naming Conventions (Armor)

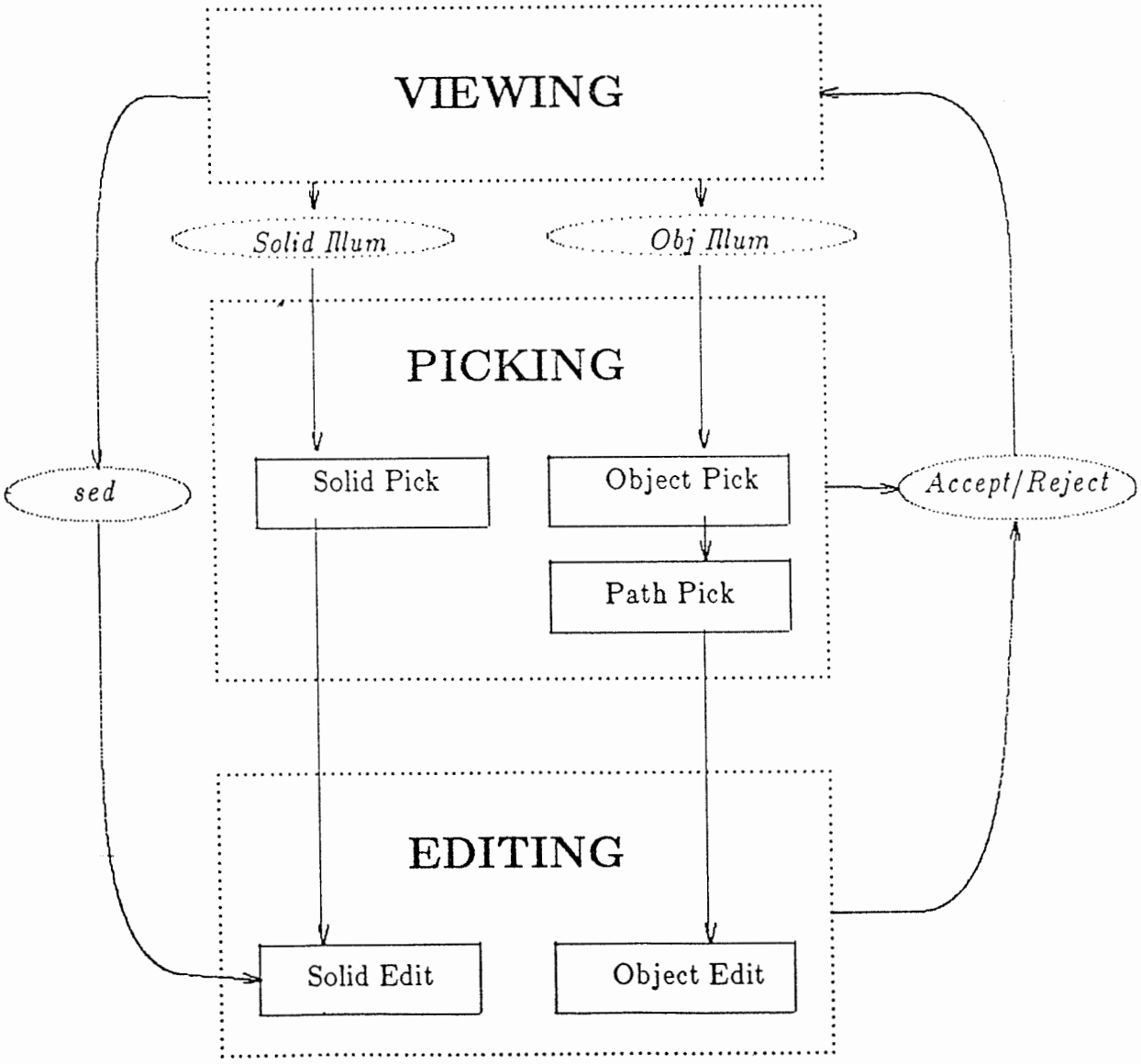
Lower Level Names	
(-1) Level	(-2) Level
turret.nn	tur.ext.nn tur.int.nn tur.air.nn —
hull.nn	hull.ext.nn hull.int.nn hull.air.nn
suspension.nn	-none-

MGED DISPLAY



MGED STATES

MGED always resides in one of 3 major states - viewing, picking, or editing. The viewing state is the default state. To perform any editing to the model, MGED must be in one of the editing states. To enter the editing state, the desired object must be selected for editing using "object pick" or "solid pick". The only way out of the editing state is to accept or to reject the editing (even if none were done).

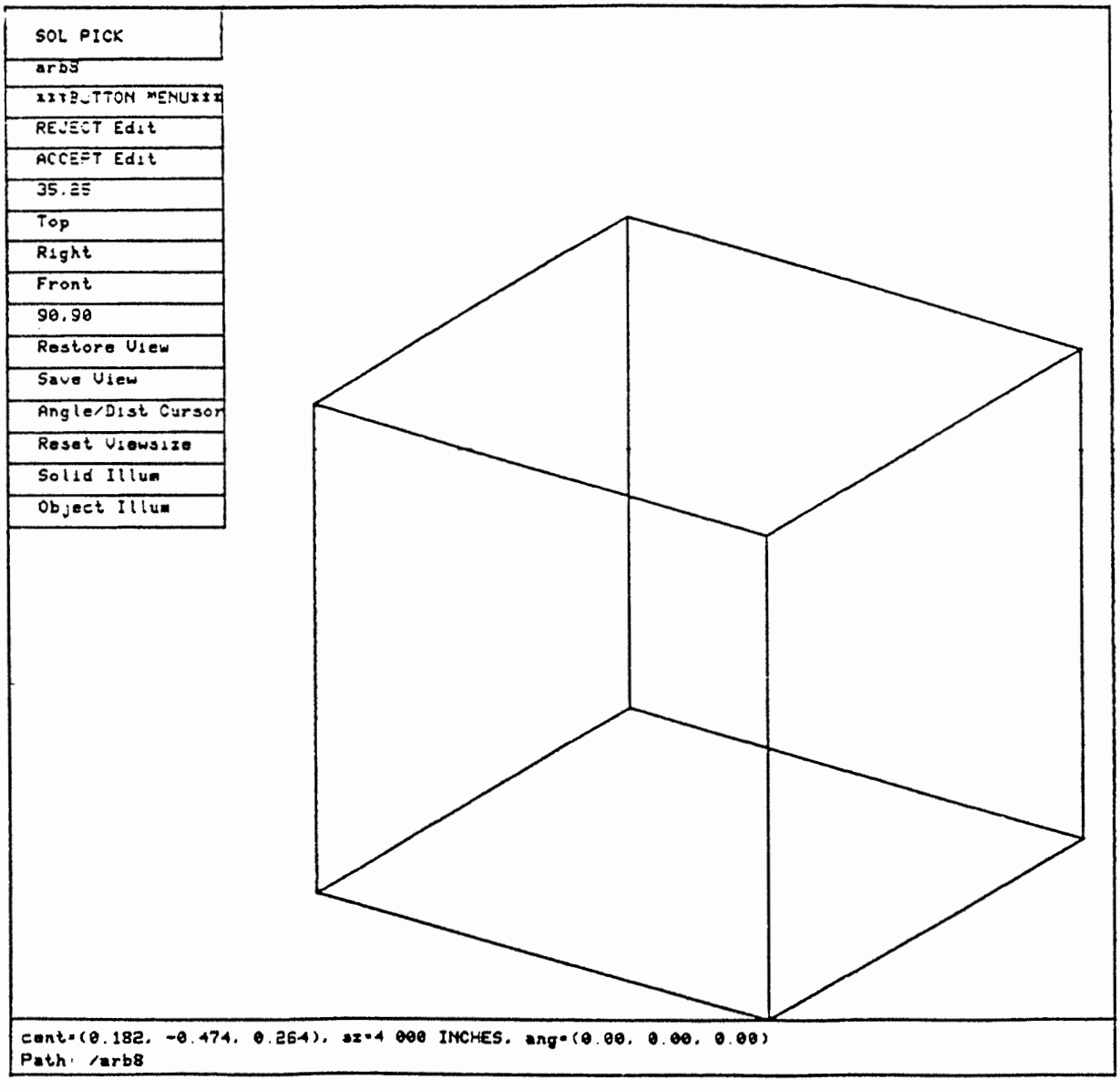


SOLID ILLUMINATE

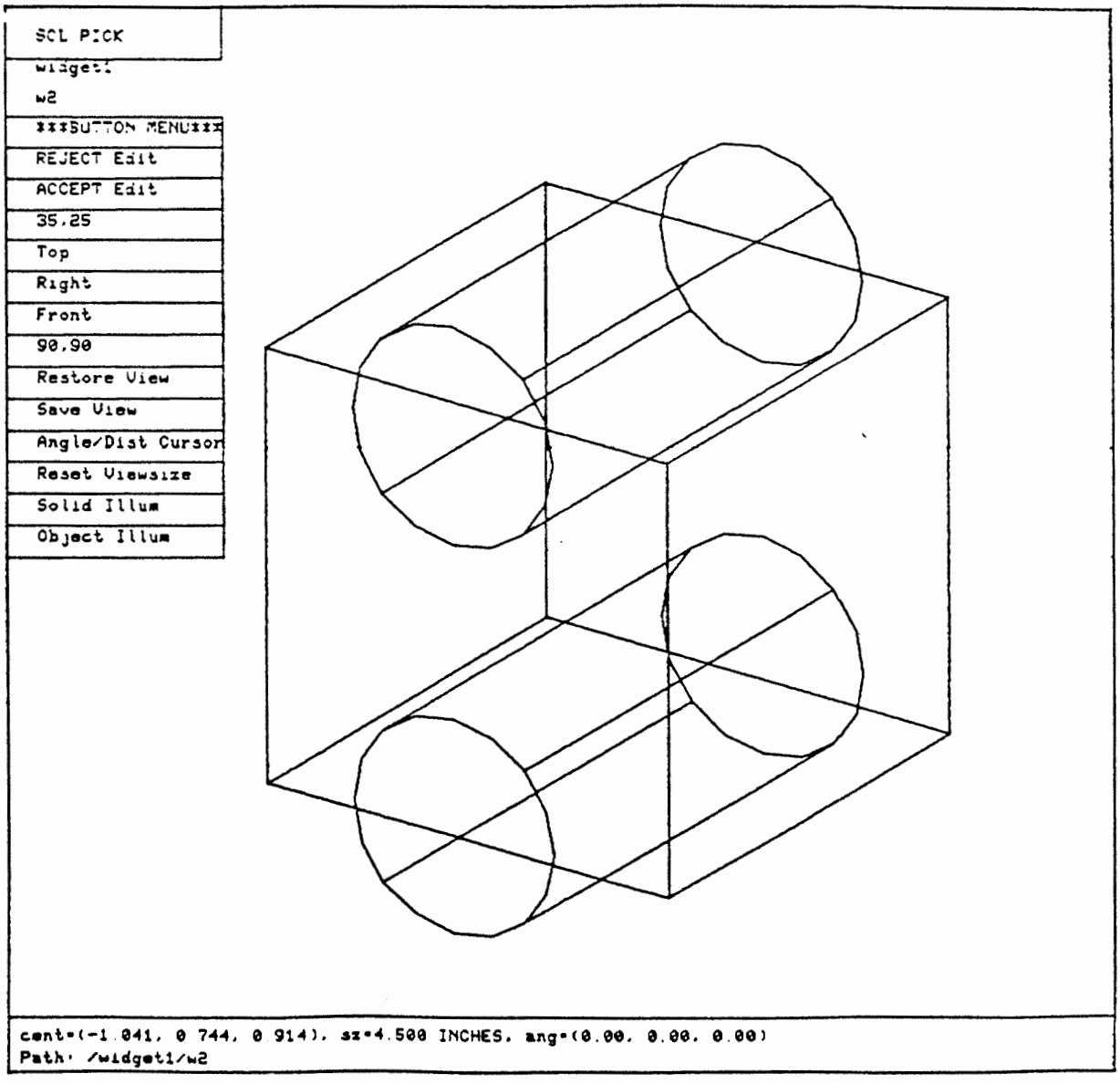
A solid must be displayed before it can be picked for editing. To pick a displayed solid for editing, press the "solid illum" button or select "Solid Illum" from the ***BUTTON MENU***. The data tablet and pen/mouse are then used to pick the solid. The surface of the data tablet is divided into as many horizontal sections as there are solids displayed. The displayed solid corresponding to the horizontal section the pen/mouse is located in will be "illuminated" (it will become brighter on black and white devices and white on color devices). The complete hierarchical path to reach the solid is also listed on the display. When the pen/mouse is pressed, MGED enters the solid edit state with the illuminated solid as the solid to be edited.

If the solid is not multiply referenced, entering "sed solidname" on the keyboard will immediately put MGED in the solid edit state with "solidname" as the edited solid.

Sample Display During Solid Illuminate



Sample Display During Solid Illuminate

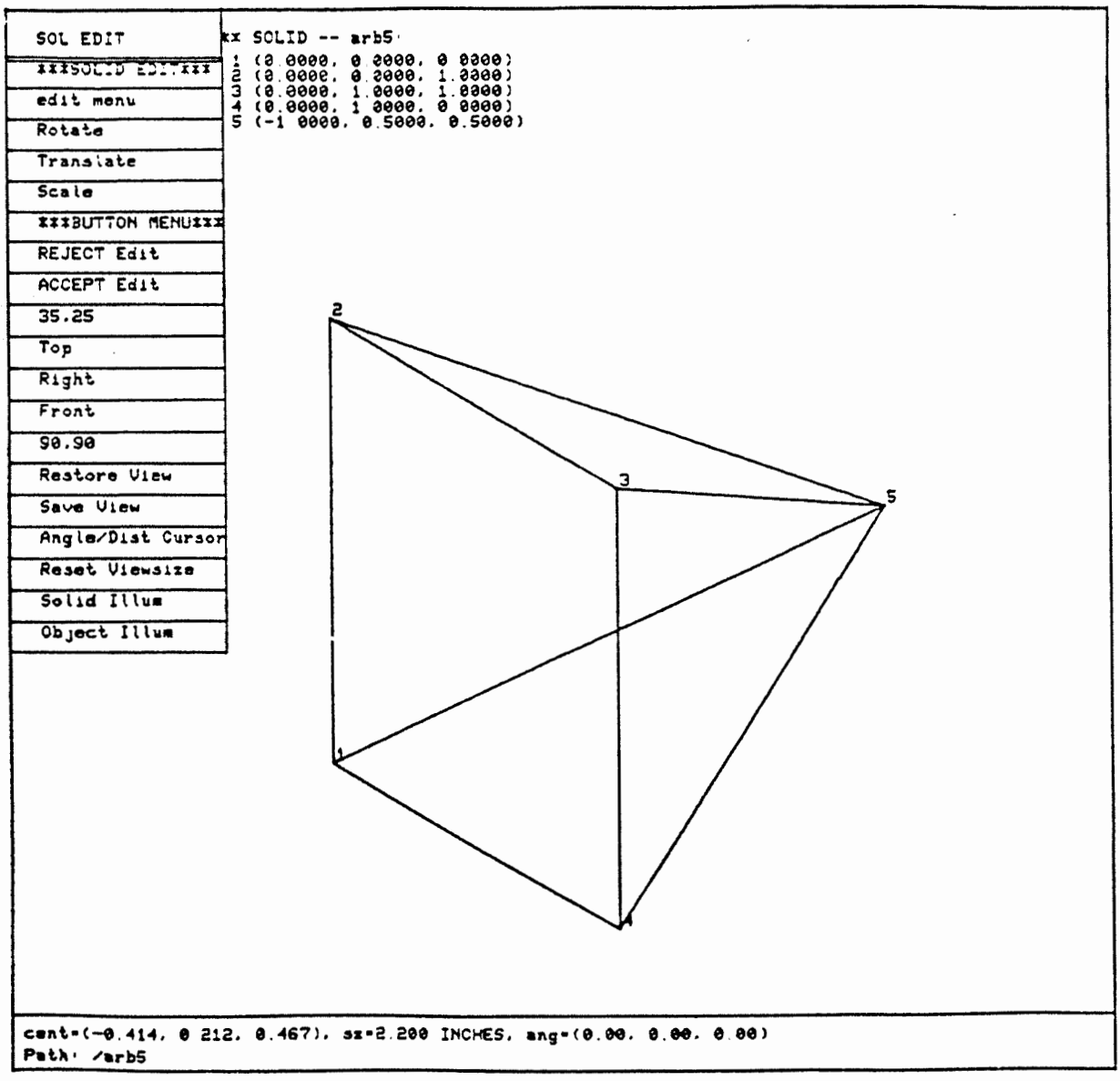


SOLID EDIT STATE

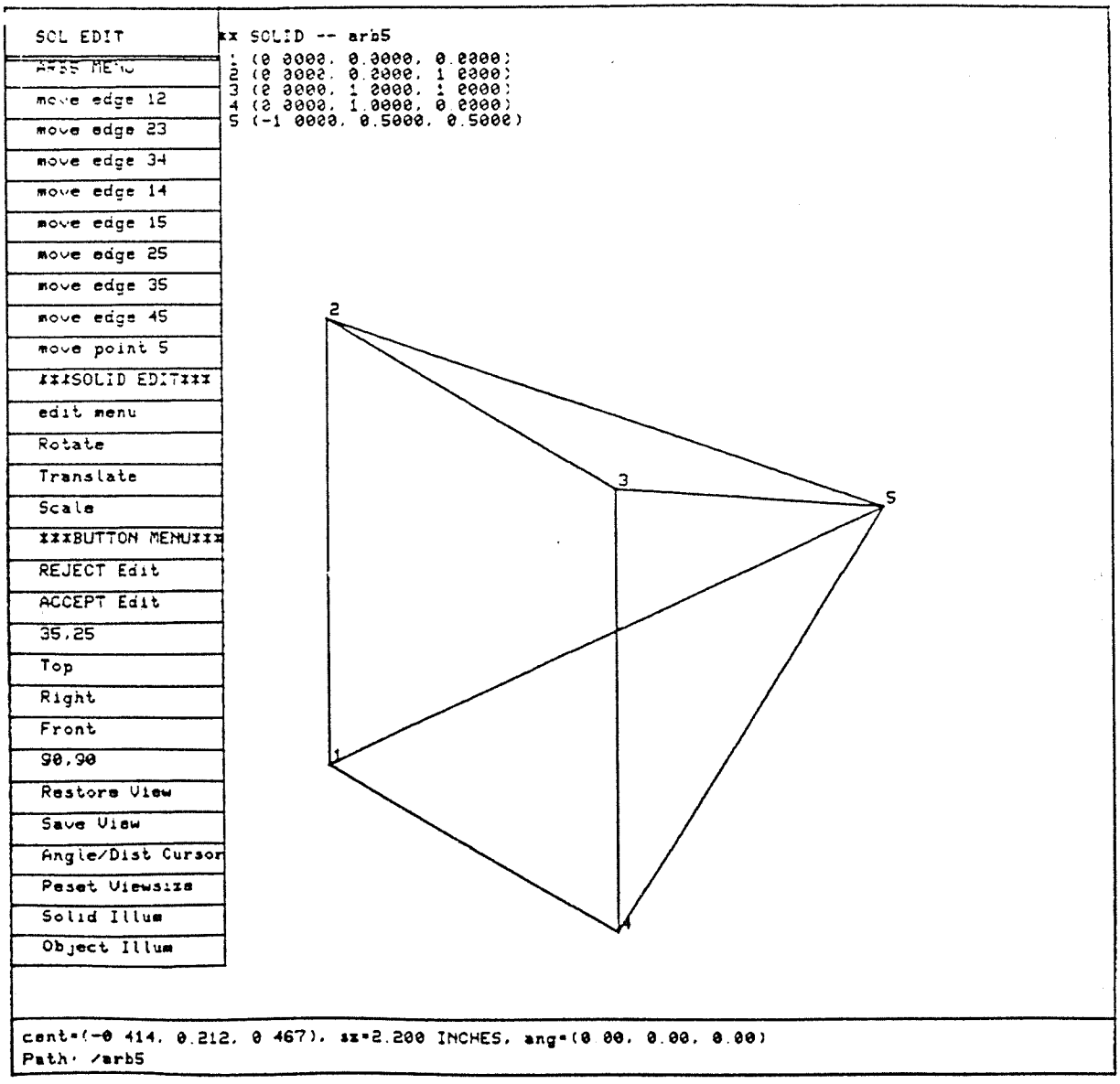
When MGED enters the solid edit state, the following occurs:

1. the edited solid remains illuminated
2. the edited solid's parameters are labeled
3. the edited solid's parameters are listed
   (and continually updated)
4. the ***SOLID EDIT*** menu is displayed
5. the parameter edit menu is initially displayed (default)

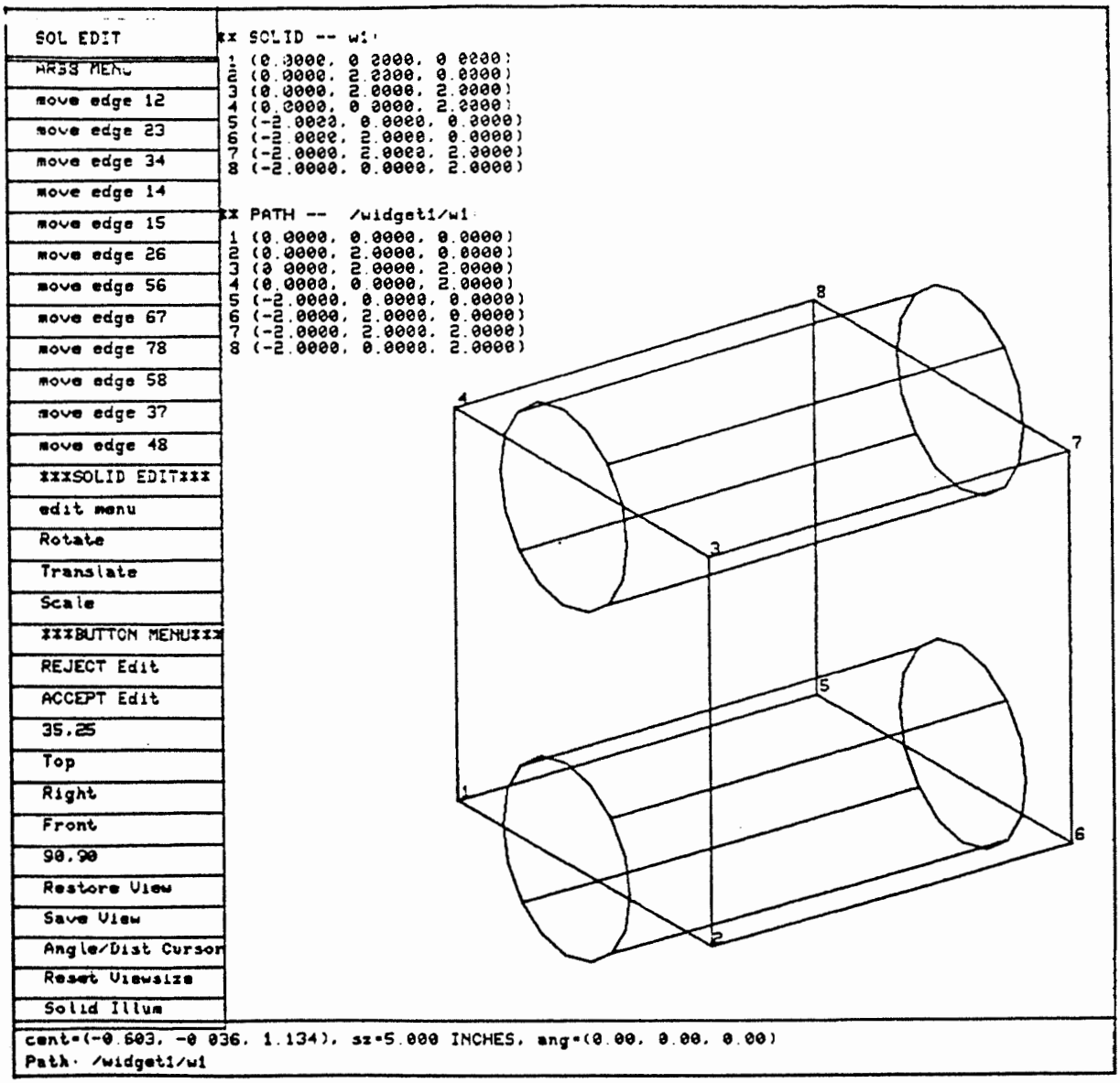
Sample Display
Solid Edit State



Sample Display
Solid Edit State



Sample Display
Solid Edit State



SOLID EDIT

ROTATE
TRANSLATE
SCALE
PARAMETERS

SOLID ROTATION

To select solid rotation, press the solid rotate button or select "Rotate" from the ***SOLID EDIT*** menu. The joy stick or appropriate rotation knobs then will rotate the edited solid about the coordinate axes. The solid is rotated about its vertex. The parameter (p) command can be used to make precise rotation changes. The values entered after the p are absolute - the rotations are applied to the solid as it existed when solid rotation was first selected. Thus entering "p 0 0 0" will "undo" any rotations performed since solid rotation was selected. The rotation about the z-axis is done first, then the y, then the x.

SOLID TRANSLATION

To select solid translation, press the solid translate button or select "Translate" from the ***SOLID EDIT*** menu. When the pen/mouse is pressed, the vertex of the edited solid will move to that location. The parameter (p) command can be used to translate the solid to a precise location. Entering "p x y z" will place the vertex of the edited solid at (x, y, z).

SOLID SCALE

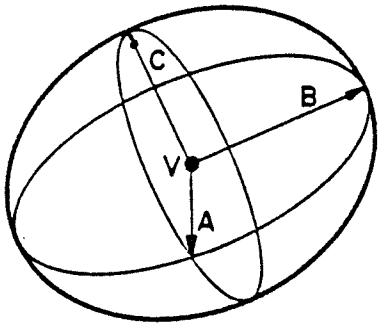
To select solid scale, press the solid scale button or select "Scale" from the ***SOLID EDIT*** menu. When the pen/mouse is pressed, the edited solid is scaled by an amount proportional to the distance the pen/mouse is from the center of the screen. If the pen/mouse is above the center, the edited solid will become larger. If it is below the center, the solid will become smaller. The parameter (p) command can be used to enter precise scale factors. The value entered is applied to the solid as it existed when solid scale was entered. Hence entering "p 1" will return the solid to its size when solid scale session first started.

Parameter Editing

To modify individual solid parameters, press the menu button or select "edit menu" from the ***SOLID EDIT*** menu. A menu listing what parameter editing is available for that particular solid type will be displayed. Using the pen/mouse select the desired item(s) from this menu. For most of the parameter editing, the "p" command can be used to make precise changes. Parameter editing is the default edit mode entered when MGED first enters the solid edit state.

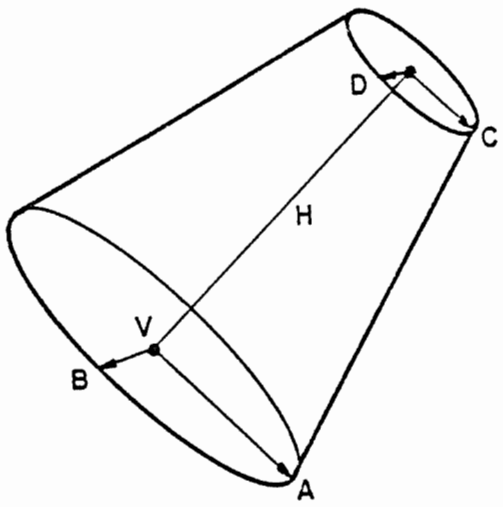
ELLG
Parameter Editing

ELLIPSOID MENU
scale A
scale B
scale C
scale A,B,C



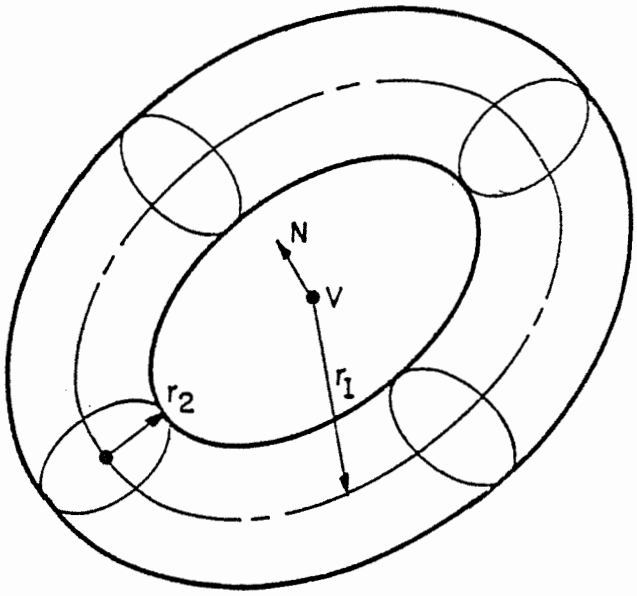
TGC
Parameter Editing

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 AxB
move end H(rt)
move end H



TOR
Parameter Editing

TORUS MENU
scale radius 1
scale radius 2



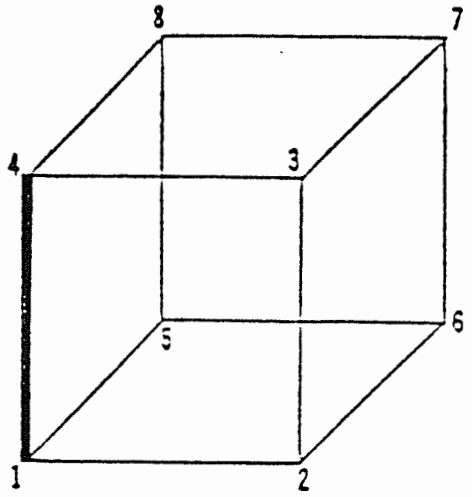
ARB
Parameter Editing

There are three classes of editing that can be done to ARBs....move edges, move faces, and rotate faces. There is an "ARB control" menu (see below) from which one selects the type of parameter editing to be done. A specific ARB edit menu will appear dependent on which parameter editing option was selected. The "return" entry on each menu will return the "ARB control" menu to the screen.

ARB Control Menu
move edges
move faces
rotate faces

Moving ARB Edges

Face vertices of an ARB must lie in a plane, thus in general one cannot move individual vertices. The following scheme for moving edges has been implemented to allow for editing of ARB vertices.



MOVE EDGE 14 .....

1. define line 14 which contains edge 14
2. find bounding planes: 125(6) & 347(8)
3. identify planes which will change: 145(8) & 123(4)
4. define "new line 14" so it contains desired point
5. find new pt 1 (1') by intersecting new line 14 with plane 125(6)
6. find new pt 4 (4') by intersecting new line 14 with plane 347(8)
7. define the new "changed" planes: 1'4'5 and 1'4'2
8. move pt 8 into new plane 1'4'5
9. move pt 3 into new plane 1'4'2

Using this scheme, all face vertices remain planar.

Moving ARB Faces

To move an ARB face, select the desired face from the "move face" menu. A point is then "input" either through a pen press or through the "p" command. The plane containing the edited face is then moved so that it contains the input point. The new face is then calculated and the ARB is displayed.

Rotating ARB Faces

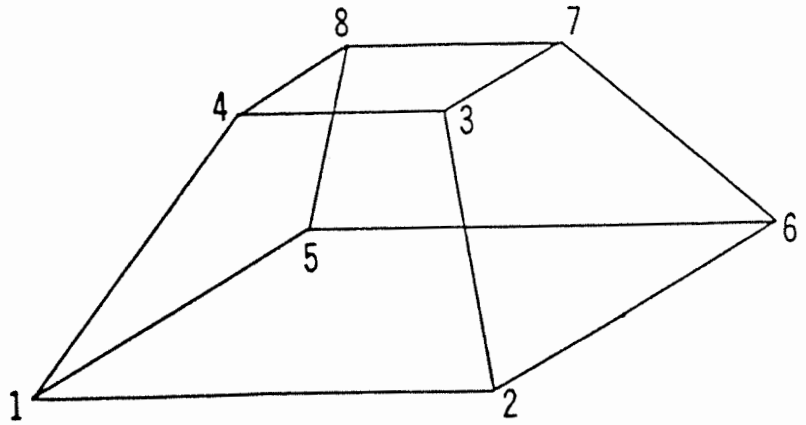
ARB faces may be rotated around any of the vertices comprising that face. First, select the desired face from the "rotate face" menu. You will then be asked to select the vertice number around which to rotate the face. The face can be rotated about the three coordinate axes. The knobs (Rotate X, Rotate Y, and Rotate Z) are used for this purpose. For precise rotations, use the "p" command. If three values are entered after the "p", then they are interpreted as angles (absolute) of rotation about the X, Y, Z axes respectively. If only two values are entered, then they are considered as rotation and fallback angles for the normal to that face. The "eqn" command can also be used here to define the plane equation coefficients of the face being rotated.

ARB8
Parameter Editing

ARB8 EDGES
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
RETURN

ARB8 FACES
move face 1234
move face 5678
move face 1584
move face 2376
move face 1265
move face 4378
RETURN

ARB8 FACES
rotate face 1234
rotate face 5678
rotate face 1584
rotate face 2376
rotate face 1265
rotate face 4378
RETURN

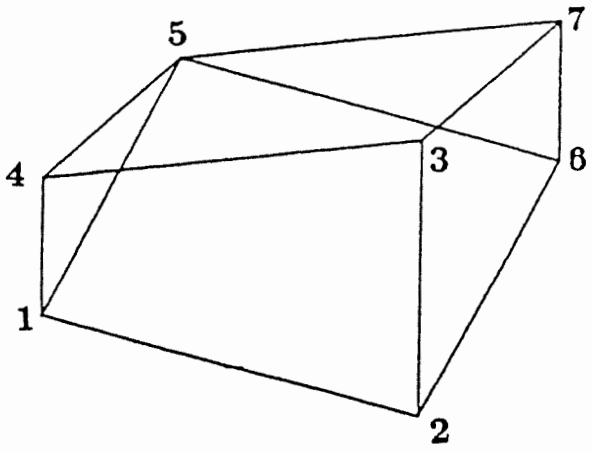


ARB7
Parameter Editing

ARB7 EDGES
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 37
move edge 57
move edge 45
move point 5
RETURN

ARB7 FACES
move face 1234
move face 2376
RETURN

ARB7 FACES
rotate face 1234
rotate face 567
rotate face 145
rotate face 2376
rotate face 1265
rotate face 4375
RETURN

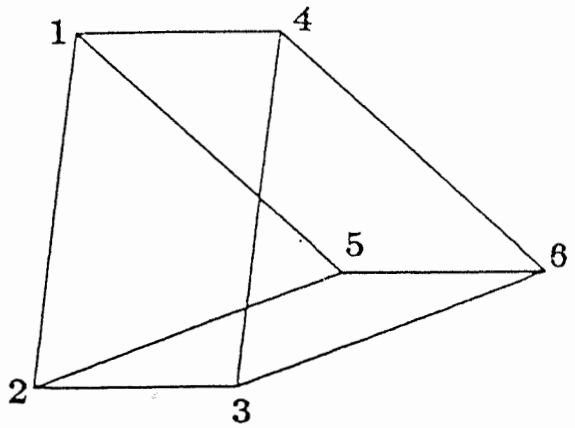


ARB6
Parameter Editing

ARB6 EDGES
move edge 12
move edge 23
move edge 34
move edge 14
move edge 15
move edge 25
move edge 36
move edge 46
move point 5
move point 6
RETURN

ARB6 FACES
move face 1234
move face 236
move face 1564
move face 125
move face 346
RETURN

ARB6 FACES
rotate face 1234
rotate face 236
rotate face 1564
rotate face 125
rotate face 346
RETURN

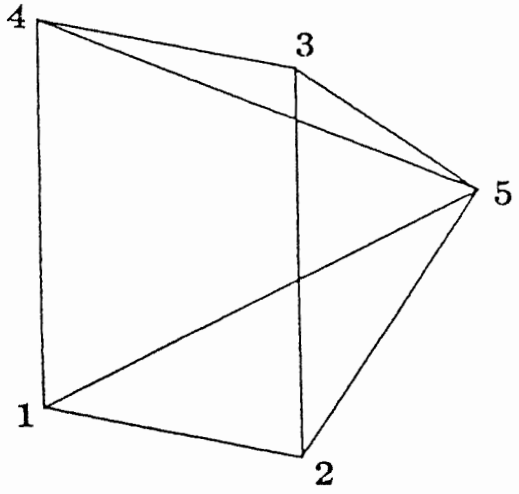


ARB5
Parameter Editing

ARB5 EDGES
move edge 12
move edge 23
move edge 34
move edge 14
move edge 15
move edge 25
move edge 35
move edge 45
move point 5
RETURN

ARB5 FACES
move face 1234
move face 125
move face 235
move face 345
move face 145
RETURN

ARB5 FACES
rotate face 1234
rotate face 125
rotate face 235
rotate face 345
rotate face 145
RETURN

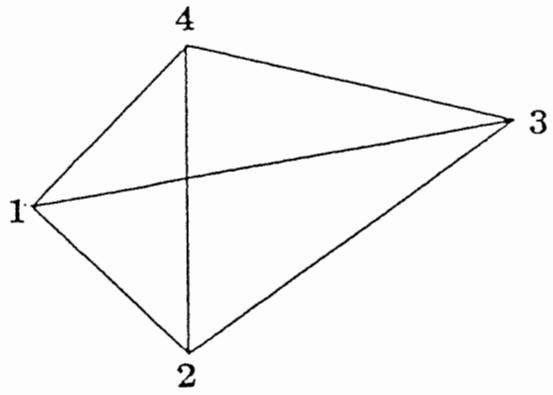


ARB4
Parameter Editing

ARB4 POINTS
move point 1
move point 2
move point 3
move point 4
RETURN

ARB4 FACES
move face 123
move face 124
move face 234
move face 134
RETURN

ARB4 FACES
rotate face 123
rotate face 124
rotate face 234
rotate face 134
RETURN



OBJECT ILLUMINATE

To select a displayed object for editing, press the object illuminate button or select "Object illum" from the ***BUTTON MENU***. The object selection is a two step process.

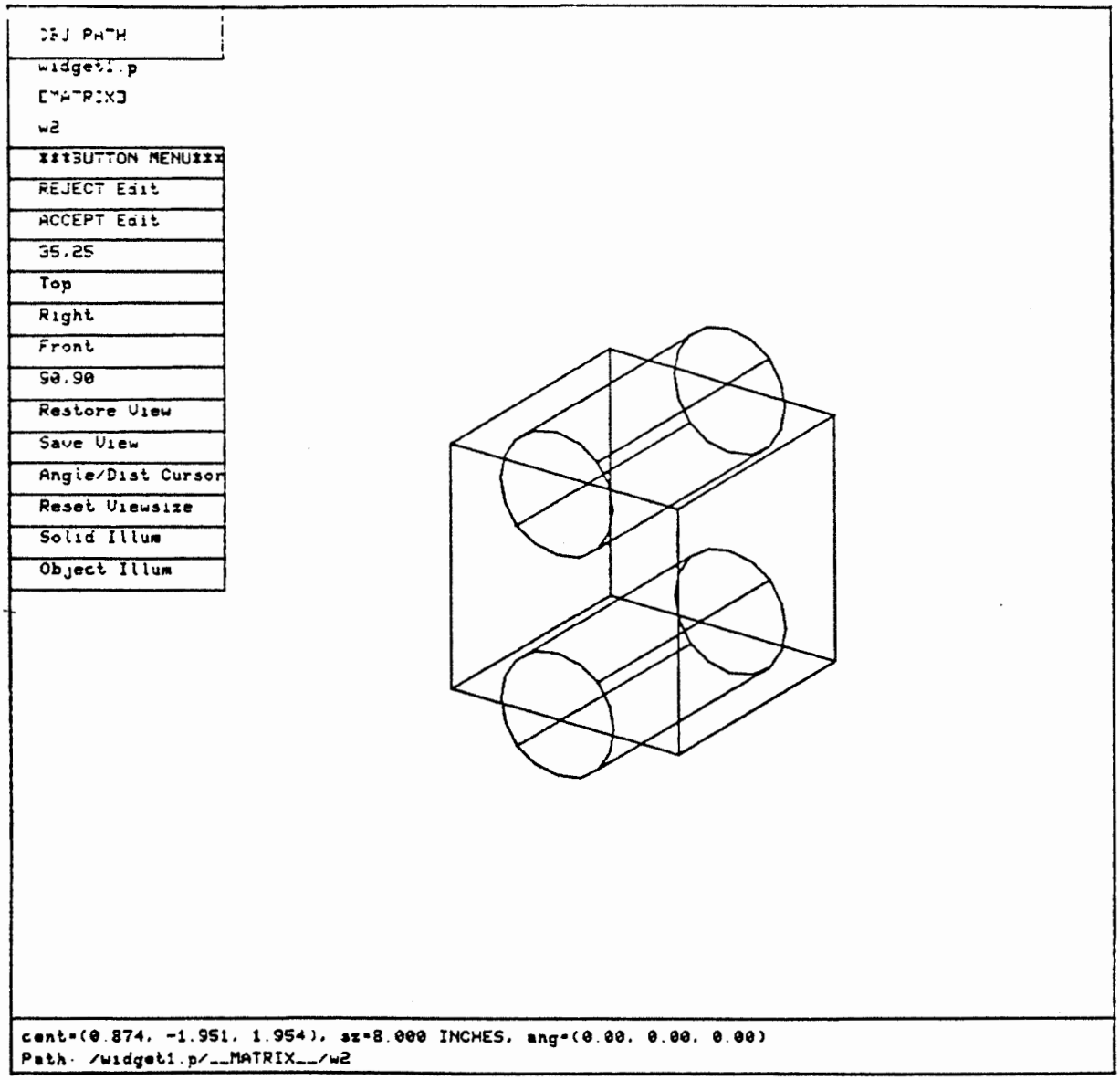
Whenever an object is displayed (using the "e" command), all paths in the object's hierarchy are traversed recursively, accumulating the transformation matrices. When the bottom of the path (a solid) is encountered, the accumulated transformations are applied to the solid's parameters and the solid is drawn. Thus every solid displayed is really a path ending with that solid. If the object has been displayed using the "E" command, the same procedure is followed, but only until a region is encountered. Then all members of the region have the accumulated transformations applied and the region is then "evaluated" and drawn.

In the first step of the object selection process, the path is selected. Again, the data tablet is divided in as many horizontal sections as there are paths drawn. The path (solid or evaluated region) corresponding to the horizontal section the pen/mouse is located in will be illuminated (brighter on B/W displays and white on color displays). This complete path is also listed on the display. When the pen/mouse is pressed the illuminated path is selected.

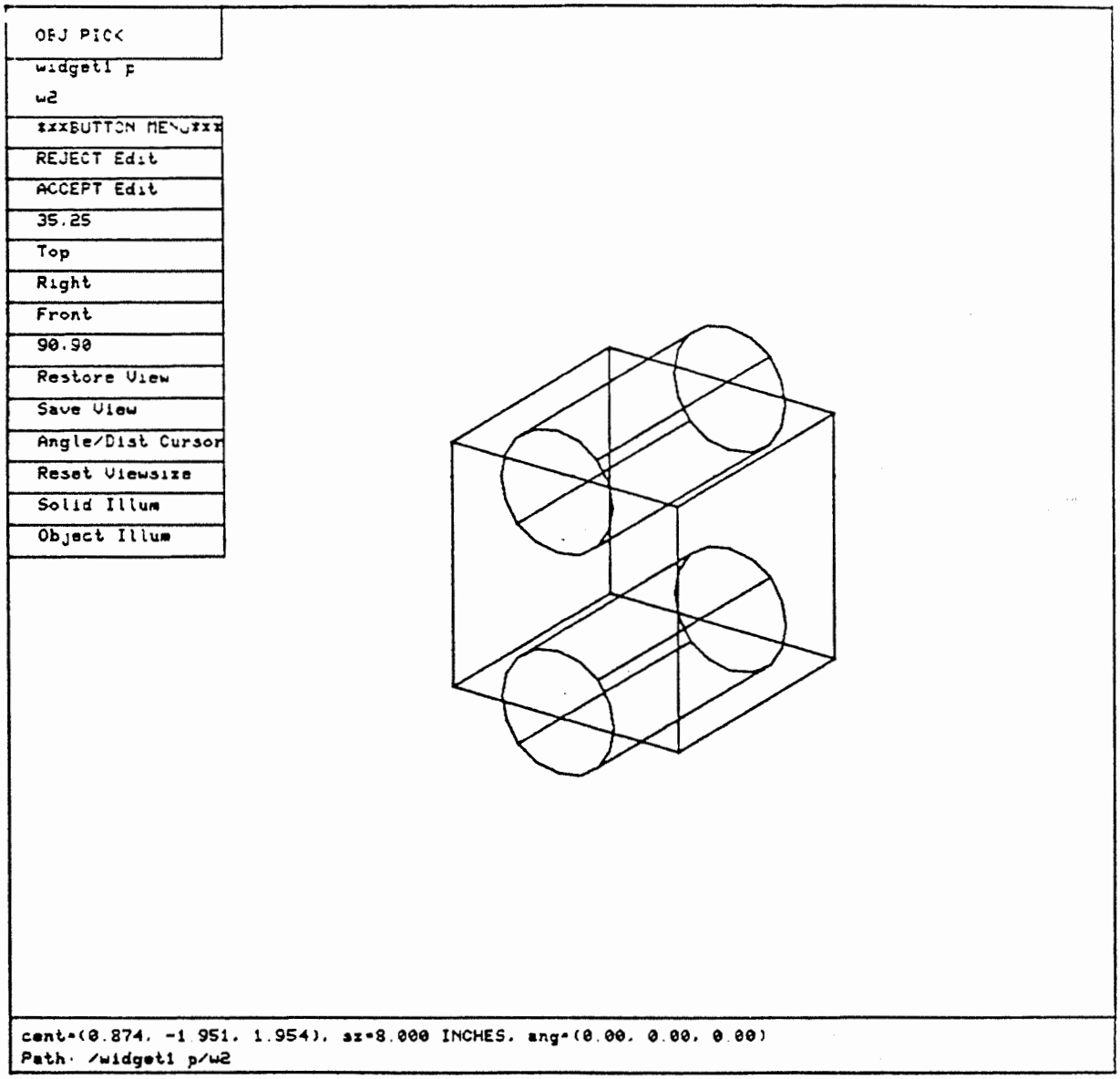
In the second step, a member of the selected path is chosen. All editing will then be applied to this member. The tablet is divided into as many horizontal sections as there are path members. The word "[MATRIX]" is used to illuminate path members and will appear above the member corresponding to the location of the pen/mouse. Pressing the pen/mouse when the desired path member is "illuminated" will put MGED in the object edit state. The editing will be performed on the path member selected.

If a solid is located at the bottom of this path, it becomes the key solid and its vertex becomes the key point. If an evaluated region is at the bottom of the path, the center of this region becomes the key point. All object editing is done with respect to this key point.

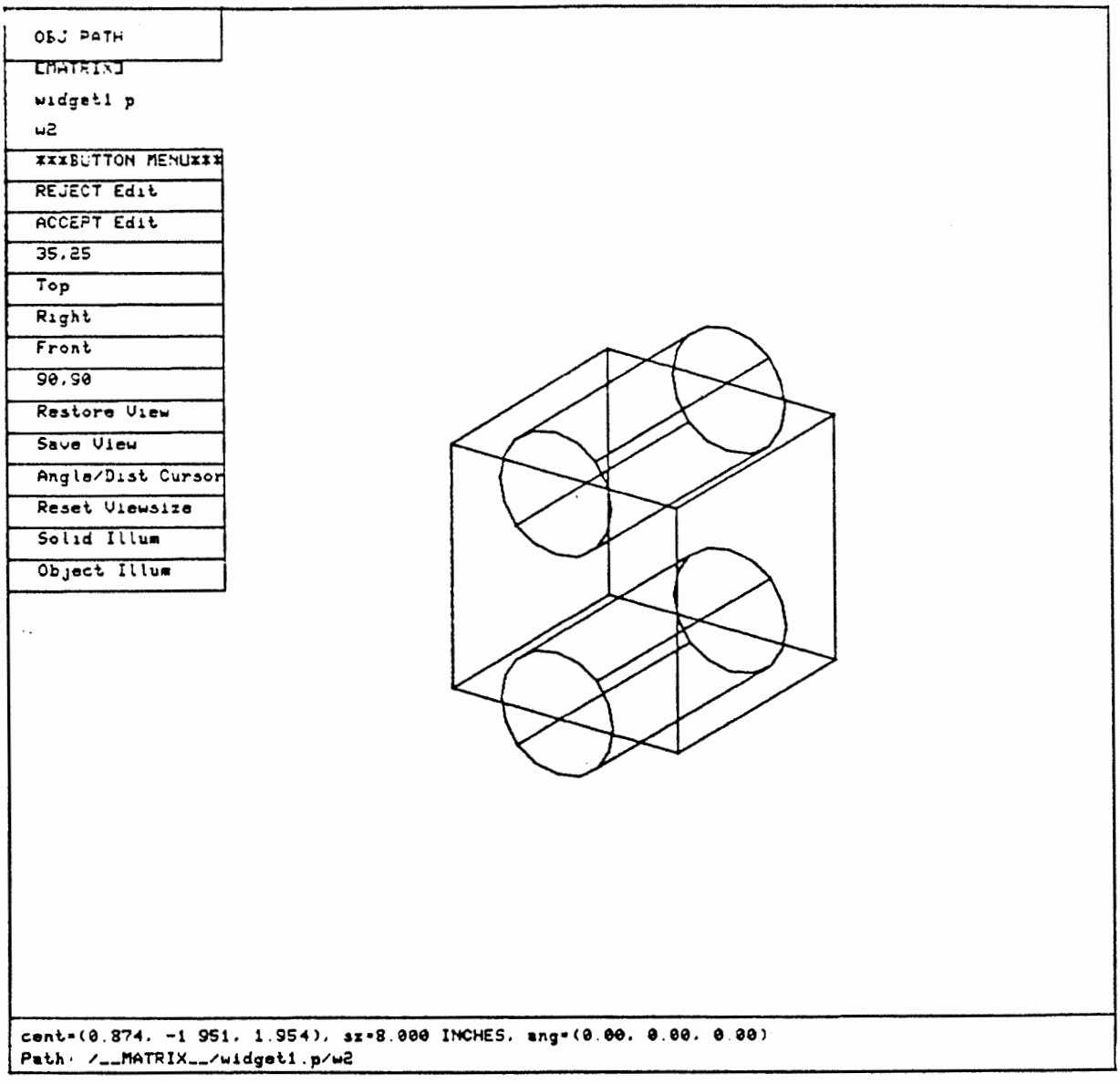
Sample Display During Object Illuminate



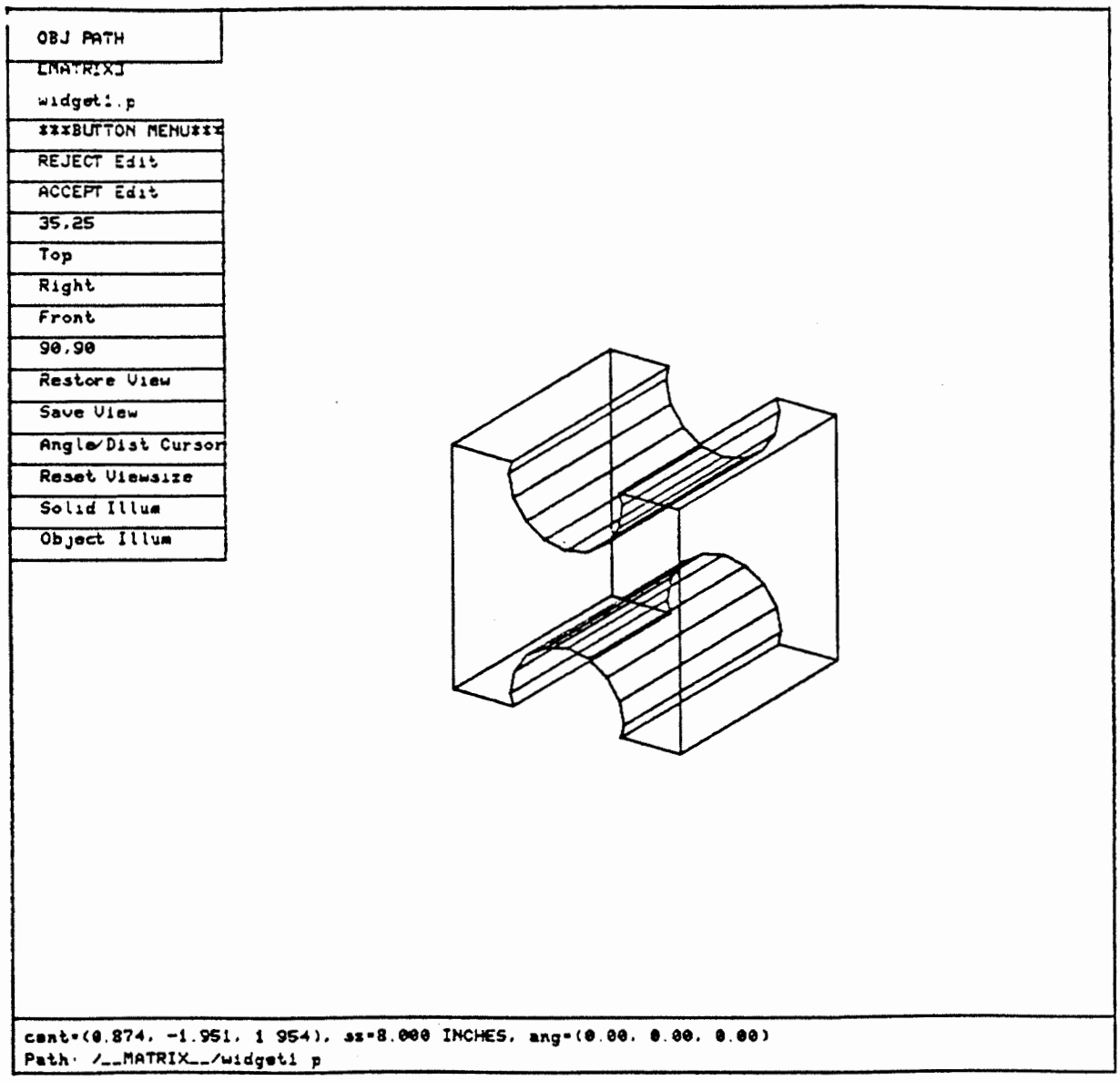
Sample Display During Object Illuminate



Sample Display During Object Illuminate



Sample Display
During Object Illuminate

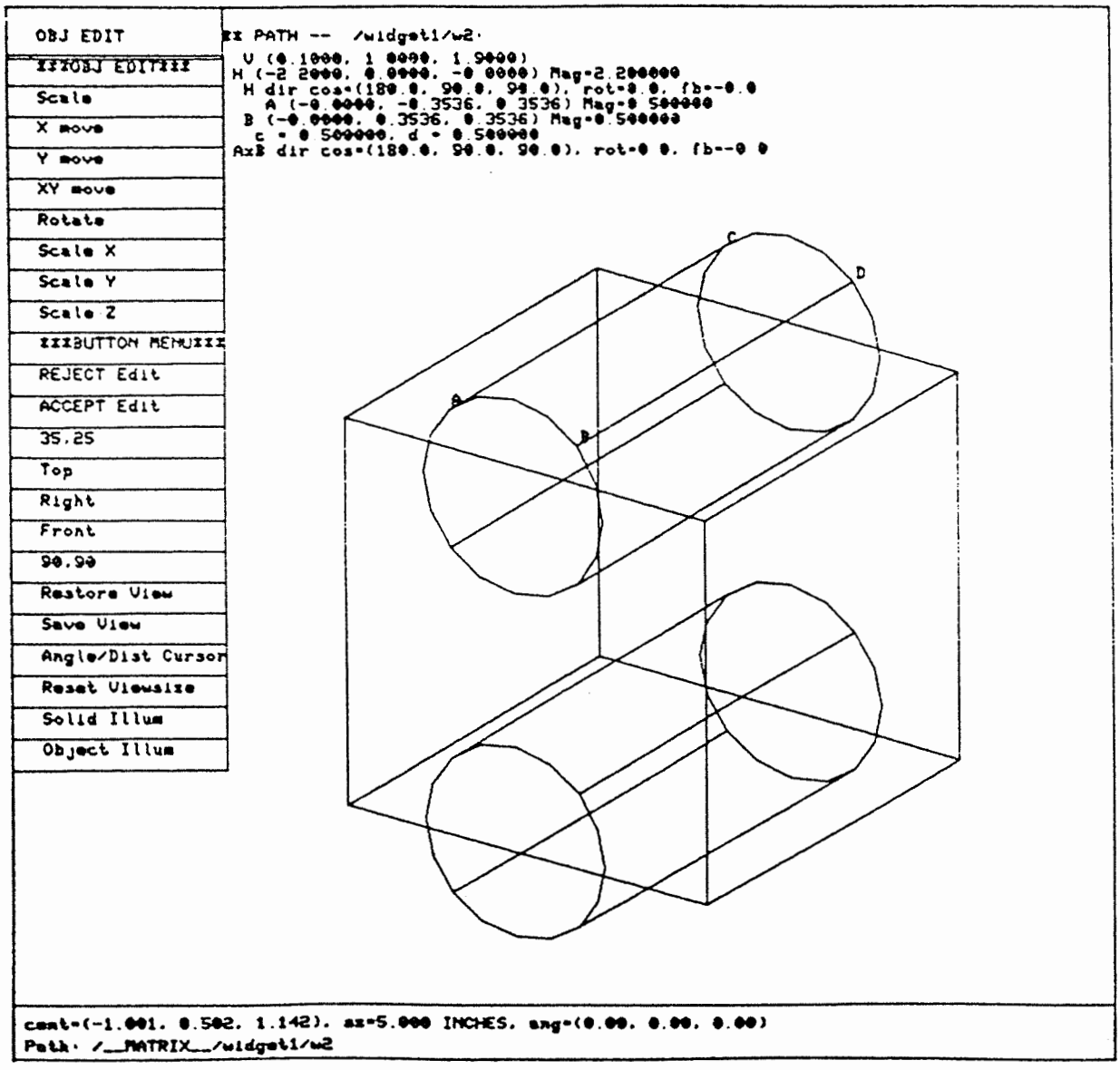


OBJECT EDIT STATE

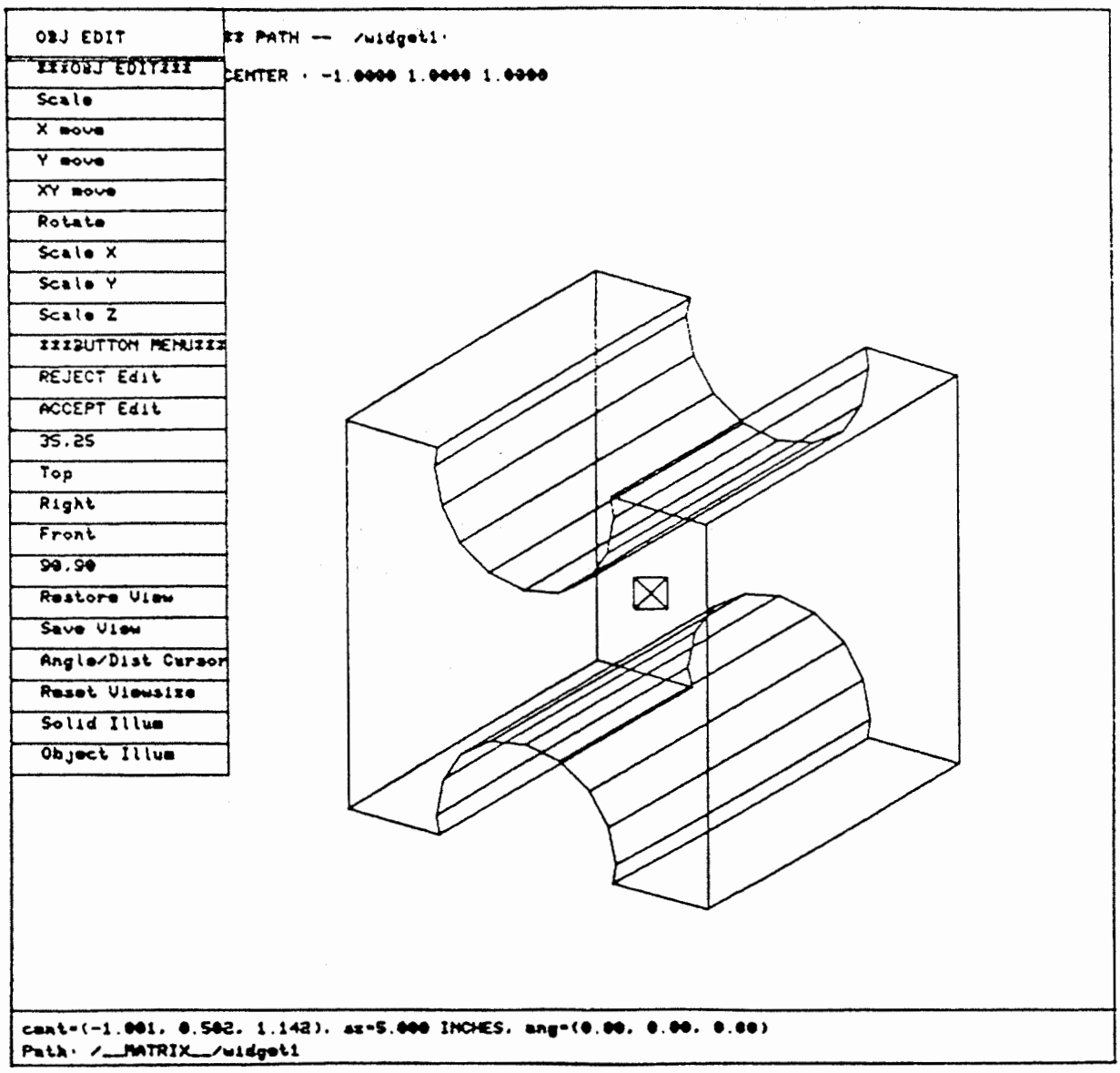
When MGED enters the object edit state, the following occurs:

1. all the solids/evaluated regions of the edited object become illuminated
2. the key solid's parameters are labeled OR the center of the key evaluated region is marked
3. the key solid's parameters are listed and continually updated OR the key evaluated region's center is listed and continually updated
4. the ***OBJ EDIT*** menu is displayed

Sample Display
Object Edit State



Sample Display
Object Edit State



OBJECT EDIT

ROTATE
TRANSLATE
GLOBAL SCALE
LOCAL SCALE

OBJECT ROTATION

To select object rotation, press the object rotate button or select "Rotate" from the ***OBJ EDIT*** menu. The joy stick or appropriate rotation knobs then will rotate the edited object about the coordinate axes. The edited object is rotated about the designated key point for this object. The "rotobj" command can be used to make precise rotation changes. The values entered after rotobj are absolute - the rotations are applied to the object as it existed when object rotation was first selected. Thus entering "rotobj 0 0 0" will "undo" any rotations performed since object rotation was selected. The rotation about the z-axis is done first, then the y, then the x.

OBJECT TRANSLATION

There are 3 ways to translate an object ... translate in the screen X direction only (X move), translate in the screen Y direction only (Y move), or just straight translation (XY move). In all cases, the complete object is translated so that the "key point" is positioned at the desired location. The "translate" command is used to enter a precise location (x,y,z) for the key point. Entering "translate x y z" will move the complete object so that the key point will be at coordinates (x,y,z).

GLOBAL OBJECT SCALE

To select global object scale, press the object scale button or select "Scale" from the ***OBJ EDIT*** menu. When the pen/mouse is pressed, the edited object is scaled about the key point by an amount proportional to the distance the pen/mouse is from the center of the screen. If the pen/mouse is above the center, the edited object will become larger. If it is below the center, the object will become smaller. The "scale" command can be used to enter precise scale factors. The value entered is applied to the object as it existed when object scale was entered. Hence entering "scale 1" will return the object to its size when the object scale session first started.

LOCAL OBJECT SCALE

Local object scaling is allowed about any of the coordinate axes. To select local scaling, press one of the buttons (OBJ Scale X, OBJ Scale Y, or OBJ Scale Z) or select "Scale X", "Scale Y", or "Scale Z" from the ***OBJ EDIT*** menu. When the pen/mouse is pressed, the edited object is scaled in the selected coordinate axis only, about the key point. The amount of scaling is proportional to the distance the pen/mouse is from the center of the screen. If the pen/mouse is above the center, the edited object will become larger in the selected axis direction. If it is below the center, the object will become smaller in the selected axis direction. The "scale" command can be used to enter precise scale factors. The value entered is applied to the object as it existed when local object scale was entered. Hence entering "scale 1" will return the object (in the selected axis direction) to its size when the object scale session first started.

KEYBOARD

COMMANDS

---

e obj1 obj2 ... objn

This command allows one to display objects on the screen. In this case, "obj1" thru "objn" will be displayed on the screen. String matching is allowed for the arguments.

E obj1 obj2 ... objn

This command is the same as the "e" command, except the regions will be evaluated before being displayed. String matching is allowed for the arguments.

B obj1 obj2 ... objn

This command is the same as the "e" command, except that the screen is cleared (Zap) before the objects are displayed. String matching is allowed for the arguments.

---

l obj1 obj2 ... objn

This command is used to list information about objects in the data file. The information listed depends on what type of record "object" is. If "object" is a combination record, then the members are listed. If "object" is a solid record, then the MGED general solid type and the parameters as presently in the data file are listed. String matching is allowed for the arguments. Note: only the solid parameters as they exist in the solid record are listed, no transformation matrix is applied. Hence, if the solid was edited as a member of a combination, the "l" command will not reflect the editing in the listed parameters.

Examples:

```
l hull turret
l turtop.s
l arb8
l s10?
```

---

analyze <solid>

This command produces information about a solid(all except ars). The information includes surface area(s) and volume. Also, in the case of arbs, edge lengths and rot and fallback angles and plane equations are given for each face. If "solid" is present that solid name will be used and analyzed. If "solid" is absent, the solid at the bottom of the present path being edited will be analyzed. String matching is allowed for the argument "solid".

---

cp oldobj newobj

This command is used to produce a copy of an object (solid or comb). In this case, the object "oldobj" will be copied into an object called "newobj". The object is copied "in place" and is not displayed on the screen.

Examples:

```
cp arb8 hullbot.s
cp tgc wheelrim.s
cp torso.r driver_torso
cp proto.man driver
```

---

cpi oldtgc newtgc

This command is a specialized copy command and is designed to be used when one is "running wires" in a description. The object being copied must a cylindrical solid (TGC). The following occurs when cpi is used: first the cylinder ("oldtgc") is copied to "newtgc"; then "newtgc" is translated to the end of "oldtgc"; then "newtgc" is displayed; and finally, MGED is put in the SOLID EDIT state with "newtgc" as the edited solid.

---

i object combname

This command is used to make an instance of an object. An instance of an object is produced by creating a combination and making the object a member. In this case, an instance of "object" is made by creating the combination record "combname" (if "combname" does not already exist) and adding "object" as a member. If "combname" already exists, then "object" is added as the next member.

An instance is used to refer to an object, without making actual copies of the object. Instances are useful when one is adding a certain component to a target description many times. Furthermore, any modifications to an object which has been instanced need only be done in the original (prototype) object. These modifications will then be automatically reflected in all the instances of the object.

Examples:

```
i heround he1
i heround he2
i heat heat1
i heat heat2
```

---

title newtitle

This command allows one to change the title of the target at any time. The string "newtitle" will become the new title and is limited to 72 characters including blanks.

Examples:

```
title XM89A - New version of tank
title M345 (groups are m345 and m345a)
```

---

mirror oldobj newobj axis

This command is used to create a new object which is the mirror image of an existing object about an axis. In this case, a mirror image of the object "oldobj" will created about the axis indicated by "axis" and the new object record will be called "newobj". The only acceptable values for the parameter "axis" are "x", "y", and "z".

Examples:

mirror tur.left.s tur.right.s y
mirror tur.top.s tur.bot.s z
mirror tur.front.s tur.back.s x
mirror lt_gun rt_gun y

CONSIDERATIONS:

- **When to mirror regions? groups?**

---

arb name rot fb

This command allows one to create an arb8 with desired rot and fallback angles. In this case, an arb8 with name = "name" will be created with the desired rotation angle of "rot" degrees and the fallback angle of "fb" degrees.

Examples:

arb top1.s 0 90
arb sidelt.s 90 0
arb upglacis.s 0 60

---

extrude #### distance

This command allows the user to project a face(####) of an arb being edited a normal distance to create a new arb. The value of "face" is 4 digits such as 1256. If the face is projected in the wrong direction use a negative "distance". One use for this command is for producing armor plates of a desired thickness.

Examples:

extrude 1234 20
extrude 2367 34.75
extrude 2367 -34.75

---

rm comb mem1 mem2 .... memn

This command allows one to delete members from a combination (group or region) record. In this case, members "mem1" through "memn" will be deleted from the combination "comb". String matching is allowed for the member (mem1...memn) arguments.

Examples:

rm tank hull wheels
rm region1 solid8 solid112
rm turtop.r tursidel.s tursider.s
rm all.g tur? sol*

---

make name type

This command will create a solid of a specified type. This solid will be named "name" and the solid type will be "type". The acceptable types are: arb8, arb7, arb6, arb5, arb4, tor, tgc, tec, rec, trc, rcc, ellg, ell, sph. This new solid will be drawn at the center of the screen. This command should be used create solids for editing.

---

mirface #### axis

This command allows one to mirror a face of an edited arb about an axis. This command is quite useful for adding air to a "symmetric" target.

---

summary <s|r|g>

This command will produce a summary of objects in the target file. If the options s, r, or g are entered a listing of the solids, regions, or groups will also be presented.

---

p dx [dy dz]

This is the parameter modification command and is used during solid editing to make precise changes. The actual meaning of the values typed after the "p", depend on what editing option is being performed. If one were translating a solid, then the values would be the x,y,z coordinates of the vertex of the solid.

---

release

This command releases the current display device (attaches the null device).

---

attach device

This command allows one to attach a display device (the present display device is released first). The present acceptable values for "device" are vg, mg, tek, tek4109, rat, plot, and nu. NOTE: this command should also be used to FIX a hung display instead of the fix command. The "plot" device will produce a UNIX-plot of the present view (including MGED faceplate) on a display device using a specific filter. You will be asked which filter to use. Some examples:

ikplot ..... ikonas framebuffer (low res)
ikplot -h ..... ikonas framebuffer (high res)
tplot -T110 ..... laser printer
tplot -Tmeg ..... Megatek
tplot -T4014 ..... Tek4014
tplot -Thpgl ..... HP7550A plotter

---

rotobj xdeg ydeg zdeg

This command allows one to make precise rotations of an object during object editing. MGED must be in the "object edit" state for this command to have effect. If "object rotation" is not in effect, MGED will select this option for you. The object will be rotated "xdeg" about the x-axis, "ydeg" about the y-axis, and "zdeg" about the z-axis. The rotation is "absolute"....the total rotation since the beginning of object editing will be equal to the input values. The rotation is done about the "KEY" point for the object being edited.

---

scale xxx.xx

This command allows one to make precise scaling changes during object editing. MGED must be in the "object edit" state for this command to have an effect. If one of the scaling options is not in effect, MGED will select the "global scale" option. If global scaling is in effect, the edited object will be scaled by a TOTAL amount equal to the input value. If one of the local scaling options has been selected, the object will be scaled in the selected axis direction by an amount equal to the input value. All scaling is done about the "KEY" point of the object being edited.

---

translate xxx.xx yyy.yy zzz.zz

This command allows one to make precise translation changes to an object during object editing. MGED must be in the "object edit" state for this command to have any effect. If "object translation" is not in effect, MGED will select this option for you. The "KEY" point of the object being edited will move to the input coordinates.

---

mv old new

This command is used to rename objects in the data file. In this case, the object "old" will be renamed "new". A note of caution: the name is changed only in the object record itself, not in any member records. Thus if the object "old" appears as a member of any other object, the name will not be changed there. To rename all occurrences of an object use the "mvall" command.

Examples:

mv test hull
mv g00 air
mv g1 turret

---

mvall old new

This command is used to rename all occurrences of an object in the data file. In this case, the object "old" will be renamed "new" for every occurence.

---

units ab

This command allows one to change the local or working units at ANY time. The only allowable values for "ab" are "mm", "cm", "m", "in", or "ft".

Examples:

units mm
units in

---

g group obj1 obj2 ..... objn

This command creates or appends to a combination record and is used to group objects together either for editing or displaying purposes. In this case, "obj1" through "objn" are added as members to the combination "group". If "group" does not exist, it is created and "obj1" through "objn" are added as members. String matching is allowed on the member objects (obj1...objn). NOTE: no checking if "obji" is already a member of "group".

Examples:

g shell hull turret
g tank wheels engine crew shell
g tank track
g alls's s*

---

r region op1 sol1 op2 sol2 .... opn soln

This command is used to create regions or append to regions. If "region" exists, then solids "sol1" through "soln" are added as members with "op1" through "opn" as the defining operations. If "region" does not exist, then it is created and solids "sol1" through "soln" are added as members with "op1" through "opn" as the defining operations. A region is merely a combination record with a flag set and is distinguished from other combinations (groups) since it has meaning to the COMGEOM solid modeling system. The only acceptable values for the defining operations are "+" for intersection, "-" for difference, and "u" for union.

When a region is created, the item, air, mat, and los values are set to default values. If the "regdef" command has been used, then those values will be used, otherwise the values "1000 0 100 1" will be used. To change the item and air codes use the "item" command. To change the mat and los values, use the "mater" command. The "edcodes" command is probably the easiest and fastest way to change the "identifing codes" for a region. Note: In the past all members of a region had to be solids, but recently regions have been allowed as members of regions. Hence, the names "soli" can also be regions now. Also...as in grouping, no checking for members already in a region.

Examples:

r hulltop.r + hulltop.s - hullleft.s - hullright.s
r gun + gun.s - gunin.s
r gunair + gunin.s

---

kill obj1 obj2 ... objn

This command allows one to remove objects from the file itself. Only the object records themselves are removed, any references made to these objects still will exist. To remove the references also see the "killall" command below. String matching is allowed for the arguments.

---

killall obj1 obj2 ... objn

This command will accomplish two things: first, the object[s] will be removed from the data file just as in the "kill" command; second, all references to the object[s] will also be removed. String matching is allowed for the arguments.

---

killtree obj1 obj2 ... objn

This command will remove from the file complete trees originating with obj1, obj2, ..., objn. Every object in the designated paths will be removed from the file, hence CAUTION is urged. Make sure that "killtree" is what you want to do. Using the "paths" or "tree" command on an object before "killtree" will show what objects will be killed. String matching is allowed for the arguments.

---

find obj1 obj2 ... objn

This command will find ALL references of obj1 obj2 ... objn in the file. String matching is allowed for the arguments.

---

area <tolerance>

This command finds an estimate of the presented area of all objects in the present view from that aspect. The argument "tolerance" is the tolerance for the endpoints of line segments being "equal" and is optional.

---

plot <-zclip> <-2d> <out-file> <| filter>

This command is used to produce a UNIX-plot hardcopy of the present view of the 'geometry' on the display device. The MGED faceplate will not be drawn. Some useful examples are:

ikplot ..... ikonas framebuffer (low res)
ikplot -h ..... ikonas framebuffer (high res)
tplot -Ti10 ..... laser printer
tplot -Tmeg ..... Megatek
tplot -T4014 ..... Tek4014
tplot -Thpgl ..... HP7550A plotter

---

ae az elev

This command sets the display viewing angles using the input azimuth (az) and elevation (elev) angles.

---

regdef item <air los mat>

This command allows one to change the default codes which will be given to the next region created. If the "air" code is non-zero, then the "item" code will be set to zero.

---

edgedir delta_x delta_y delta_z

This command allows one to define the direction of an ARB edge being moved. If only two arguments are input, the code assumes these to be the rotation and fallback angles for the edge. If three arguments are present, the code will use them as "deltas" to define the direction of the edge. Note that this command can be useful to find the intersection of a line (edge) with planes (faces).

---

prefix string obj1 obj2 ... objn

This command will prefix obj1, obj2, ... objn with "string". All occurences of these names will be prefixed. String matching is allowed for the objects to prefix.

---

keep file.g obj1 obj2 ... objn

This command allows one to keep the listed objects in the file "file.g". This command is useful for pulling out parts of a description. String matching is allowed for the object names.

---

tree obj1 obj2 ... objn

This command will print the tree structure for the objects listed. String matching is allowed for the object names.

---

inside

This command is used to define a solid (inside solid) such that when it is subtracted from another certain solid (outside solid), the resulting region will have desired thicknesses. To invoke this option, enter "inside". You will then be prompted for the required data. If you are in the solid edit mode, the "outside solid" will default to the solid presently being edited. If you are in the object edit mode, the "outside solid" will default to the "key solid" of the object path being edited. If you are not in an edit mode, you will then be asked for the name of the "outside solid". Next, you will be asked what name you wish to call the new "inside solid" to be calculated. Finally, you will be asked to enter the thickness[es], depending on the solid type. The "inside solid" will then be displayed on the screen. If the thickness values input are negative, then the thickness will be directed to the "outside" of the solid.

---

t <obj1 obj2 ... objn>

This is the table of contents command. If arguments are present, a list of all objects in the file matching these names will be printed. If there are no arguments, then a listing of all objects will be printed. String matching is allowed for the arguments.

---

tops

This command will search the target file hierarchy, and list all "top level" objects (objects which are not members of any other object). This command is useful to make sure objects have been grouped properly.

---

knob id value

This command is used to emulate a "knob twist". Generally this command is used for display devices which have no actual knob peripherals (eg. tek). Any non-zero number entered for "value" is converted to 1 (if "value" is greater than zero) or is converted to -1 (if "value is less than zero). The user must enter the same command with "value" equal to zero to stop the action envoked by the knob twist.

The "id" defines which knob is to be "twisted" :

x	rotates about x-axis
y	rotates about y-axis
z	rotates about z-axis
X	slew view in x direction
Y	slew view in y direction
Z	zoom the view

Examples:

```
knob x 1
knob x 0
knob Z -1
knob Z 0
```

---

sed name

This command allows one to immediately enter the solid edit mode with the solid "name" as the edited solid. The solid "name", must be displayed but not multiply referenced.

---

item region ident air

This command allows one to change the item or air code numbers of a region. If the air code ("air") is not included, a zero is assumed. To change the air code, a zero item code must be used (see second example below).

Examples:

item region1 105
item region7 0 2
item region11 129 0

---

mater region material los

This command is used to change the material and/or the los percent for a region.

Examples:

mater hull.r1 1 100
mater turret.r34 2 88

---

color low high r g b string

This command allows one to make color assignments to a range of item codes. The arguments "low" and "high" are the item ranges. The arguments "r", "g", and "b" are the red, green, and blue values respectively (the range of these numbers is generally 0-255). The argument "string" is a string describing this class of items. A blank is considered a terminator, so there can be no blanks in this string.

---

edcolor

This command allows one to edit the existing color assignments (table). The changes are made using the user's "selected" text editor.

---

prcolor

This command prints the color assignments as they presently exist.

---

flx

This command will "flx" (restart) the display device after a hardware error.

---

rt <-fxxx>

This command will produce a color shaded iamge of objects on the display. This color shaded image will appear on a framebuffer (IKONAS). The resolution of the iamge (number of rays) is equal to xxx from the "-f" option. If the -f option is absent, 50x50 ray resolution will be used.

---

ted

This command allows one to edit a solid's parameters using the text editor defined by the users's path. The solid being edited (solid edit mode) will be the one that will be "text edited".

---

ill name

This command is used to illuminate an object ... a path containing this object ("name") will be illuminated. This command is primarily used with display devices which do not have a tablet to pick objects for editing.

---

vrot xdeg ydeg zdeg

This command rotates the VIEW "xdeg" degrees about the screen x-axis, "ydeg" degrees about the screen y-axis, and "zdeg" degrees about the screen z-axis.

---

center xx.xx yy.yy zz.zz

This command moves the screen center to (xx.xx, yy.yy, zz.zz). Using this command is one way of slewing the view.

---

press button-label

This command allows one to emulate a button press and is generally used on display devices which do not have actual button peripherals. The following are the strings allowed for "button-label" and all produce the indicated view on the device screen:

top, bottom, right, left, front, rear, 90,90, 35,25

The following is a listing of the remaining acceptable strings for "button-label" and the resulting action:

reset	reset the view
save	save the present view
restore	restor the saved view
adc	display the angle-distance cursor
oill	begin object illumination (pick)
sill	begin solid illumination (pick)
oscale	object scale
ox	object translation in x direction only
oy	object translation in y direction only
oxy	object translation
orot	object rotation
sedit	put up solid parameter menu
srot	solid rotation
sxy	solid translation
sscale	solid scale
accept	accept editing done
reject	reject editing done

Examples:

press 90,90
press front
press oill
press orot
press reject

---

size xx.xx

This command sets the view size to xx.xx and is one way of zooming the display. Making the view size smaller has the effect of zooming in on the view.

---

x

This command produces a list of all objects displayed, listing the center of the object, its size, and if it is in the present view.

---

refresh

This command will send a new display list to the display device.

---

status

This is a debug command which prints the status of the current view, including all editing matrices.

---

solids file obj1 obj2 ... objn

This command will produce an ascii summary of all solids involved with objects obj1, obj2, ..., objn. This summary will be written in 'file'. In this file, all regions and solids are numbered and will match the numbers of a COMGEOM deck produced by VDECK or GIFT if the objects are entered in the same order. The file 'file' will be overwritten if it already exists. String matching is allowed for the objects.

---

regions file obj1 obj2 ... objn

This command will produce an ascii summary table of all regions involved with objects obj1, obj2,...,objn. This summary table will be written in 'file'. In this file, all regions and solids will be numbered and will match the numbers of a COMGEOM description produced by VDECK or GIFT if the objects are entered in the same order. This file will be identical to the "solids" command except the actual parameters are not listed. The file 'file' will be overwritten if it already exists. String matching is allowed for the object names.

---

idents file obj1 obj2 ... objn

This command will produce an ascii summary table of all region idents involved with objects obj1, obj2,...,objn. This summary table will be written in 'file'. In this file, all regions will be numbered and will match the region numbers of a COMGEOM description produced by VDECK or GIFT if the objects are entered in the same order. At the end of this file will be the same region information, but ordered by ident number. The file 'file' will be overwritten if it already exists. String matching is allowed for the object names.

---

dup file.g {string}

This command will compare the current object data file with another MGED file "file.g" and list any object names common to both files. If "string" is present, all object names in "file.g" will be prefixed with "string" before comparing for duplicate names. Generally, one uses "string" only when duplicate names are found without it.

---

cat file.g {string}

This command will concatenate another MGED file "file.g" onto the present object data file. If "string" is present, all names in "file.g" will be prefixed with this string. No objects from "file.g" will be added if the name already occurs in the current object file...the object names will be listed and skipped. However, this command should be used in conjunction with the "dup" command to eliminate any problems with duplicate names.

---

track

This command adds track components to the data file to fit specified "wheel" data. Solids, regions, and a group containing the regions will be created and displayed on the screen. You will be prompted for all required data.

---

paths

This command will print ALL paths matching an "input" path. You will be asked to enter the path to match. The path members are entered on ONE line, seperated by spaces. The input path need not be a complete path, but must contain at least one object. All paths with the same first members as the input path will be listed. This command is useful for finding complete paths which begin with certain objects.

---

listeval

This command will "evaluate" and list ALL paths matching an input path, including the parameters of the solids at the bottom of each path. These parameters will reflect any editing contained in the path listed. You will be asked to enter the path to match. The path members are entered on ONE line, seperated by spaces. The input path need not be a complete path, but must contain at least one object. Note that since the solid parameters are printed, this could be a rather lengthy listing depending on the completeness of the input path.

---

copyeval

This command allows one to copy an "evaluated solid", that is a complete path ending in a solid. You will be asked to enter a complete path. Again, this path is entered on ONE line with the members seperated by spaces. If you do not know the complete path, use the "paths" command above to find it. Next, you will be asked to enter the name of this copied solid. The input path will be traversed and the accumulated path transformations will be applied to the parameters of the bottom solid. These new parameters will then be the copied solid. Note: this command is useful for making "dummy solids" to subtract for overlaps between objects which have been "object" edited.

3ptarb

This command will produce a "plate mode" arb8 given 3 points on one face, 2 coordinates of the 4th point on that face, and a thickness. The 3rd coordinate of the 4th point will be solved for and the "other" face will be a normal distance (=== the desired thickness) away. You will be asked for all necessary input.

---

rfarb

This command will produce a "plate mode" arb8 given a point on one face, rotation and fallback angles for that face, 2 coordinates of the 3 remaining points on that face, and a thickness. The 3rd coordinates of the three points will be solved for, and the other face will be a normal distance equal to "thickness" away. You will be asked for all necessary input.

---

push obj1 obj2 ... objn

This command will "push" an object's path transformations to the solid's parameters located at the bottom of each path. If a conflict is encountered, then an error message is printed and NOTHING is done. A conflict occurs when the same solid "ends" different paths but the transformations are different. Conflicts could occur when "instanced" or "copied" groups occur in an object's paths or when a solid is a member of 2 regions which have been edited separately. A complication of this command, which will not be listed as a conflict, is when a solid's parameters have been changed by a push, yet this solid is referenced by another object which was NOT included in the push. The user should beware of this situation. This command can be very useful when "adding" parts from another file. Once the "object editing" of these new parts is completed, the push command will put the editing done to the solid level. Since the added parts should have no cross-references with the existing objects, there should be no problems.

---

edcodes obj1 obj2 ... objn

This command provides for an easy way to modify the code numbers (item, air, material, and los) of ALL regions found in paths beginning with an object. For each object, all paths beginning with that object are traversed until a region is encountered...at which time the following is listed:

item air mat los /obj1/.../region.n

The cursor then jumps back to "item" at the beginning of the line. At this time the cursor can be advanced only by entering a new item code (only digits are allowed) or by hitting the "space bar" or "tab key". The space and tab will move the cursor to the next position in the line. Pressing the "backspace key" will move the cursor to the beginning of the next location to the left. Moving the cursor "past" the los location will return it to the beginning of the line. When the code numbers are as you desire, a "CR" will print the next line for editing. At any time, pressing the "R" key will restore the idents on the current line to the way they were originally. A ^C (control C) or a "q" will abort the process at the current line. String matching is allowed for the object names.

---

tab obj1 obj2 ... objn

This command will produce a listing of objects as they are stored in the MGED object data file. String matching is allowed for the objects.

---

whichid item1 item2 ... itemn

This command will list ALL regions in the data file which have certain item codes.

---

in name type {parameters}

This command allows one to enter a new solid directly via the keyboard. The user will be prompted for any missing input and the solid will be displayed on the screen. WARNING: only minimal checking of parameters is done.

Z

This is the Zap command and clears all objects from the screen.

---

d obj1 obj2 ... objn

This command allows one to remove objects from the display screen. In this case "obj1" thru "objn" will be removed from the display. String matching is allowed on the arguments.

---

facedef ####

This command is used to define the face plane of an ARB that is being edited. The following is the option menu displayed when this command is used:

- **a** planar equation
- **b** 3 points
- **c** rot and fb angles + fixed point
- **d** same plane thru fixed point
- **q** quit

To select any of these methods of defining a plane, enter the appropriate letter. You will then be asked for the desired input.

---

eqn <A B C>

This command is used when one is rotating the face of an edited ARB and defines the coefficients of the face planar equation (Ax + By + Cz).

%

This command allows one to escape to the shell. To return to mged, enter a control-d.

?

This is the help command and produces a listing of all the available commands.

---

q

This is the normal quit (halt) command.

MGED COMMAND SUMMARY

Key	Argument[s]	Description
e	obj1* obj2* ... objn*	display objects on the screen
E	obj1* obj2* ... objn*	display objects evaluating regions
B	obj1* obj2* ... objn*	Zap screen, display objects
d	obj1* obj2* ... objn*	delete objects from screen
cp	oldobj newobj	copy 'oldobj' to 'newobj'
cpi	oldtgc newtgc	copy 'oldtgc' to 'newtgc' inverted
Z	-none-	Zap (clear) the screen
g	groupname obj1* obj2*....objn*	group objects
r	region op1 sol1....opn soln	create/modify a region
i	object instname	create instance of an object
mv	oldname newname	rename object
mvall	oldname newname	rename all occurences of an object
l	object*	list object information
kill	obj1* obj2* ... objn*	remove objects from the file
killall	obj1* obj2* ... objn*	remove object[s] + references from file
killtree	obj1* objn* ... objn*	remove complete paths **CAREFUL**
t	object*	table of contents
mirror	oldobj newobj axis	mirror image of an object
mirface	#### axis	mirror face #### about an axis
extrude	#### distance	extrude an arb face
item	region item air	change region item/air codes
mater	region material los	change region mat/los codes
rm	comb mem1* mem2*....memn*	delete members from combination
units	<mm|cm|m|in|ft>	change the units of the objectfile
title	new-title	change the title of the description
p	dx [dy dz]	precise commands for solid editing
rotobj	xdeg ydeg zdeg	rotate(absolute) an edited object
scale	factor	scale(absolute) an edited object
translate	x y z	translate an edited object
arb	name rot fb	make arb8 with rot and fb
analyze	<solid*>	print much info about a solid
summary	<s|r|g>	solid/region/group summary
tops	-none-	list all top level objects
find	obj1* obj2* ... objn*	find all references to an object
area	<endpoint-tolerance>	find presented area of E'd objects
plot	<-zclip> <-2d> <out-file> <| filter>	make UNIX-plot of view
color	low high r g b str	assign color(r g b) to item range
edcolor	-none-	text edit the color/item assignments
prcolor	-none-	print the current color/item assignments
make	name type	create & display a primitive
fix	-none-	restart the display after hangup
rt	[options]	raytrace view onto framebuffer

* (string matching allowed for this argument)

MGED COMMAND SUMMARY (CONTINUED)

Key	Argument[s]	Description
release	-none-	release current display processor
attach	<nu|tek|tek4109|plot|mg|vg|rat>	attach new display processor
ae	az elev	rotate view using azimuth & elevation angles
regdef	item <air los mat>	set default codes for next region created
ted	-none-	text edit a solids parameters
vrot	xdeg ydeg zdeg	rotate view
ill	name	illuminate object
sed	solidname	solid edit the named solid
center	x y z	set view center
press	button-label	emulate button press
knob	id value	emulate knob twist
size	value	set view size
x	-none-	debug list of objects displayed
status	-none-	print view status
refresh	-none-	send new control list
edcomb	comb flag item air mat los	edit comb record info
edgedir	delta_x delta_y delta_z	define direction of an ARB edge being moved
in	name type {parameters}	type-in a new solid directly
prefix	string obj1* obj2* ... objn*	prefix objects with 'string'
keep	file.g obj1* obj2* ... objn*	keep objects in 'file.g'
tree	obj1* obj2* ... objn*	list tree for objects
inside	--prompted for input--	find inside solid
solids	file obj1* obj2* ... objn*	make ascii solid parameter summary in 'file'
regions	file obj1* obj2* ... objn*	make ascii region summary in 'file'
idents	file obj1* obj2* ... objn*	make ascii region ident summary in 'file'
edcodes	obj1* obj2* ... objn*	edit region ident codes
dup	file {prefix}	checks for dup names in 'file' & current file
cat	file {prefix}	cat's 'file' onto end of current file
track	--prompted for input--	builds track given appropriate 'wheel' data
3ptarb	--prompted for input--	makes arb8 given 3 pts, etc.
rfarb	--prompted for input--	makes arb8 given point, rot, fallback, etc
whichid	ident1 ident2 ... identn	list all regions with given ident
paths	--prompted for input--	lists all paths matching input path
listeval	-prompted for input--	gives 'evaluated' path summary
copyeval	--prompted for input--	copy an 'evaluated' path-solid
tab	obj1* obj2* ... objn*	list objects as stored in data file
push	obj1* obj2* ... objn*	push object transformations to solids
facedef	#### {data}	define plane of an edited ARB face
eqn	<A B C>	define plane coefficients of rotating ARB face
q	-none-	quit
%	-none-	escape to shell
?	-none-	help message

* (string matching allowed for this argument)

CONVERSION

CODES

BINARY <===> ASCII

Since MGED files are binary, they must be converted to ascii in order to be used on a machine of different architecture. Two conversion codes are available for this purpose: g2asc and asc2g. The procedure is to first convert the binary MGED file to ascii on the resident machine; second, send this ascii file to the target machine; and finally convert back to binary before using on the target machine. The following is an example:

on the resident machine....

```
g2asc < file.g > file.asc
```

{"send" file.asc to target machine}

on the target machine....

```
asc2g < file.asc > file.g
```

You are now able to use file.g on the target machine.

GIFT COMGEOM TO MGED

Two steps are required to convert a GIFT COMGEOM formatted file to the MGED file structure. There are 2 possible formats for the GIFT COMGEOM decks - GIFT4 and GIFT5. The conversion codes "cvt4" and "cvt5" convert GIFT4 and GIFT5 files to an old version of the MGED files. The conversion code "cv" will then convert to the present MGED file structure.

In the conversion process, names are needed for all the object records created. The solids from the COMGEOM formatted file are given names "si" where "i" is the position of the solid in the solid table. The regions are handled in the same way. In addition, cvt4 and cvt5 make groups out of regions according to their item numbers. As an option, both cvt4 and cvt5 will add a suffix to the solid and region names.

The cv code will convert old MGED files to the present form. The cv code will ask for the units and a title for the old MGED file. It is important that the units be correct.

EXAMPLE:

```
cvt5 m93.cg5 m93.vg .m93
cv m93.vg m93.g


cvt4 m45.cg4 m45.vg
cv m45.vg m45.g
```

All these conversion codes are located in /vld/keith/Bin.

MGED TO GIFT COMGEOM

The code which produces a GIFT COMGEOM deck from an MGED file is called VDECK. The code is interactive and will give a menu of commands when a ? is entered. The user builds a list of objects from the MGED file table of contents which are to be converted to a COMGEOM deck. To add an object to this list use the i[nsert] command. To actually produce the COMGEOM deck, enter d[eck]. Three files will be created: solids, regions, region_ids. These files when combined produce a GIFT5 COMGEOM formatted file.

EXAMPLE:

```
vdeck m55.g
      i component
      d
      q
cat solids regions region_ids > m55.cg5
rm solids regions region_ids
```

*U.S. GOVERNMENT PRINTING OFFICE:1987-180-991:60078

USER EVALUATION OF REPORT

Please take a few minutes to answer the questions below; tear out this sheet, fold as indicated, staple or tape closed, and place in the mail. Your comments will provide us with information for improving future reports.

1. BRL Report Number _____

2. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.)

_____

_____

_____

3. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.) _____

_____

_____

4. Has the information in this report led to any quantitative savings as far as man-hours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate.

_____

_____

5. General Comments (Indicate what you think should be changed to make this report and future reports of this type more responsive to your needs, more usable, improve readability, etc.) _____

_____

_____

_____

6. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

Name: _____

Telephone Number: _____

Organization Address: _____

_____

_____

FOLD HERE

Director
US Army Ballistic Research Laboratory
ATTN: DRDAR-BLA-S
Aberdeen Proving Ground, MD 21005

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 12062 WASHINGTON, DC
POSTAGE WILL BE PAID BY DEPARTMENT OF THE ARMY

Director
US Army Ballistic Research Laboratory
ATTN: DRDAR-BLA-S
Aberdeen Proving Ground, MD 21005

FOLD HERE