Geomview Manual

Geomview Version 1.9.5

for Unix

February 2009

Mark Phillips et al.

Copyright © 1992-1998 The Geometry Center
Copyright © 1998-2006 Stuart Levy, Tamara Munzner, Mark Phillips
Copyright © 2006-2007 Claus-Justus Heine

Introduction to Geomview

Geomview is an interactive program for viewing and manipulating geometric objects, originally written by staff members of the Geometry Center at the University of Minnesota, starting in 1991. It can be used as a stand-alone viewer for static objects or as a display engine for other programs which produce dynamically changing geometry. It runs on many kinds of Unix computers, including Linux, SGI, Sun, and HP. It also runs with Cygwin. This manual describes Geomview version 1.9.

Geomview is free software, available under the terms of the GNU Lesser Public License; see [Copying], page 3, for details.

Geomview and this manual are available for download from `http://www.geomview.org`.

Permission is granted to make copies of this manual.

If you have questions or comments about Geomview or this manual, consider joining in the `geomview-users` mailing list, which is a forum in which users of Geomview communicate to answer each others' questions and to share news about what they are doing with Geomview. The Geomview authors participate in this list and sometimes post answers to questions there. To join the list, visit the list web page at `http://lists.sourceforge.net/mailman/listinfo/geomview-users`.

Distribution

Geomview is *free software*; this means that everyone is free to use it and free to redistribute it on certain conditions. Geomview is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of Geomview that they might get from you. The precise conditions are found in the GNU Lesser Public License that comes with Geomview and also appears following this section.

One way to get a copy of Geomview is from someone else who has it. You need not ask for our permission to do so, or tell any one else; just copy it. If you have access to the Internet, you can get the latest distribution version of Geomview at `http://www.geomview.org`.

You may also receive Geomview when you buy a computer. Computer manufacturers are free to distribute copies on the same terms that apply to everyone else. These terms require them to give you the full sources, including whatever changes they may have made, and to permit you to redistribute the Geomview received from them under the usual terms of the General Public License. In other words, the program must be free for you when you get it, not just free for the manufacturer.

Copying

NOTE: Geomview is distributed under the GNU LESSER GENERAL PUBLIC LICENSE. For the purposes of this license we think of Geomview as if it were a "library", and of Geomview external modules as "programs that link with the library". We do this because we specifically want to allow proprietary programs and modules to use Geomview.

GNU LESSER PUBLIC LICENSE

Version 2.1, February 1999

Copyright © 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program

by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. The modified work must itself be a software library.
   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply

to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices. ^L Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

   b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

   c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

   d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

   e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components

(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the library's name and a brief idea of what it does.
Copyright (C) <year>  name of author

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library `Frob' (a library for tweaking knobs) written by James
Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

History of Geomview's Development

Geomview was originally written at the Geometry Center at the University of Minnesota in Minneapolis. The Geometry Center was a research and education center funded by the National Science Foundation, with a mission to promote research and communication of mathematics. Much of the work there involved the use of computers to help visualize mathematical concepts.

The project that eventually led to Geomview began in the summer of 1988 with the work of Pat Hanrahan on a viewing program called MinneView. Shortly thereafter Charlie Gunn begin developing OOGL (Object Oriented Graphics Language) in conjunction with MinneView. Many people contributed to OOGL and MinneView, including Stuart Levy, Mark Meuer, Tamara Munzner, Steve Anderson, Mario Lopez, Todd Kaplan.

In 1991 the staff of the Geometry Center began work on a new improved version of OOGL, and a new and improved viewing program, which they called Geomview. At that time essentially the only game in town for interactive 3D graphics was Silicon Graphics (SGI), so Geomview was developed initially on SGI workstations, using IRIS GL. The first version was finished in January of 1992. It immediately became very popular among visitors to the Geometry Center, and through the Center's ftp archive (this was before the web) people at other institutions began using it too.

In addition to SGI workstations the Geometry Center had quite a few NeXT stations, so soon after Geomview was running on SGIs the staff developed a version for NeXTStep as well. By this time there were several thousand people using it around the world.

A few years later the staff ported Geomview to X windows and OpenGL, and eventually, with the demise of NeXT, the NeXT version fell by the wayside.

In its mission to foster communication among researchers and educators, the Geometry Center developed a web site, www.geom.umn.edu, in late 1993. It was one of the first 300 web sites in existence. A part of the web site was of course devoted to Geomview, and helped to spread the word about its existence.

The Geometry Center closed its "brick and mortar" facilities in August of 1998 (NSF cut its funding), but the web site continued to exist, and Geomview continued to be very popular around the world. In December of 1999 some of the former Geometry Center staff set up `http://www.geomview.org` as a permanent home on the web for Geomview.

Geomview's original authors, as well as a number of other volunteers around the world, are still actively involved in using and developing Geomview.

Authors

Tamara Munzner, Stuart Levy, and Mark Phillips are the original authors of Geomview. Celeste Fowler, Charlie Gunn, and Nathaniel Thurston also made significant contributions. Daniel Krech and Scott Wisdom did the NeXTStep and RenderMan port, and Daeron Meyer and Tim Rowley did the port to X windows. Many other Geometry Center staff members, as well as several people elsewhere, also contributed.

Mark Phillips wrote this manual, with substantial help from Stuart Levy and Tamara Munzner. Countless Geomview users have also been of great help by reading it and pointing out mistakes.

Supported Platforms

Geomview 1.9 should – in principle – compile and run on any fairly recent Unix-like operating system. Specifically, it runs on Linux and on Cygwin (Cygwin emulates a SystemV Unix-like environment under Microsoft Windows). Unluckily Geomview compiles with MacOS X (Darwin), but seemingly communicating with Geomview by means of pipes and sockets cause segmentation faults. Feel free to fix that! See [Contributing], page 151, for details.

How to Pronounce “Geomview“

The word 'Geomview' is a combination of the first syllable of the word 'geometry', and the word 'view'. The authors pronounce it with the accent on the first syllable

GE-om-view

Some people put the accent on the second syllable, where it falls in the word 'geometry', but the original authors, who invented the name, prefer the accent-on-first-syllable pronunciation.

1 Overview

Geomview's main purpose is to display objects whose geometry is given, allowing interactive control over details such as point of view, speed of movement, appearance of surfaces and lines, and so on. Geomview can handle any number of objects and allows both separate and collective control over them.

The simplest way to use Geomview is as a stand-alone viewer to see and manipulate objects. It can display objects described in a variety of file formats. It comes with a wide variety of example objects, and you can create your own objects.

You can also use Geomview to handle the display of data coming from another program that is running simultaneously. As the other program changes the data, the Geomview image reflects the changes. Programs that generate objects and use Geomview to display them are called *external modules*. External modules can control almost all aspects of Geomview. The idea here is that many aspects of the display and interaction parts of geometry software are independent of the geometric content and can be collected together in a single piece of software that can be used in a wide variety of situations. The author of the external module can then concentrate on implementing the desired algorithms and leave the display aspects to Geomview. Geomview comes with a collection of sample external modules, and this manual describes how to write your own.

Geomview is the product of an effort at the Geometry Center to provide interactive geometry software that is particularly appropriate for mathematics research and education. In particular, Geomview can display things in hyperbolic and spherical space as well as Euclidean space.

Geomview allows multiple independently controllable objects and cameras. It provides interactive control for motion, appearances (including lighting, shading, and materials), picking on an object, edge or vertex level, snapshots in SGI image file or Renderman RIB format, and adding or deleting objects is provided through direct mouse manipulation, control panels, and keyboard shortcuts.

Geomview supports the following simple data types: polyhedra with shared vertices (.off), quadrilaterals, rectangular meshes, vectors, and Bezier surface patches of arbitrary degree including rational patches. Object hierarchies can be constructed with lists of objects and instances of object(s) transformed by one or many 4x4 matrices. Arbitrary portions of changing hierarchies may be transmitted by creating named references.

Geomview can display 3-D graphics output from Mathematica and Maple.

2 Tutorial

This chapter leads you through some of the basics of using Geomview. Work through this chapter in front of a computer where you can try out the examples given here to get a feel for what you can do with Geomview.

To start Geomview, login to the computer and get a shell window. A shell window is a window in which you can type Unix commands; the prompt in the window usually ends with a '%'. In the shell window (the mouse cursor must be in the window) type the following (`Enter` here means hit the "Enter" key):

```
geomview tetra dodec Enter
```

This command starts up Geomview and loads two example objects, a tetrahedron and a dodecahedron. After a few seconds three windows should appear; see Figure 2.1.

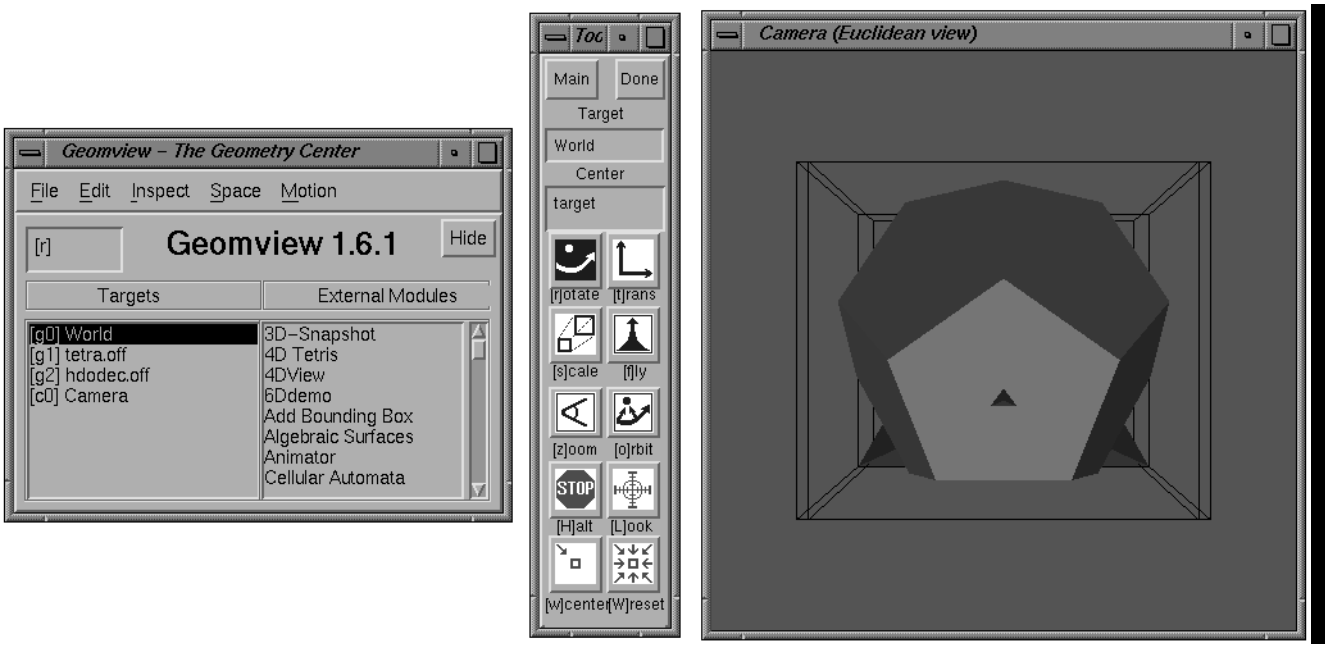


Figure 2.1: Initial Geomview display.

The panel on the left is Geomview's main control panel; it's called the *Main* panel. The skinny panel in the middle is the *Tools* panel and is for selecting different kinds of motions. The window on the right is the camera window and in it you see a large tetrahedron and a dodecahedron which is partially obscured by the tetrahedron.

Geomview has lots of panels but by default it displays only these three. We'll describe some aspects of these and a couple of the others in this tutorial. You can read more about these and other panels in the later chapters of this manual.

Put the mouse cursor in the camera window and press down and hold the left mouse button. Now, while holding down the button, slowly move the mouse around. You should see the picture rotate in the direction you move the mouse. If you lift up on the mouse button while moving the mouse, the picture continues rotating. To stop it, hold the mouse very still and click down and up on the left mouse button.

Geomview uses the *glass sphere* model for mouse-based motion. This means you are supposed to think of the object as being inside an invisible sphere and the mouse cursor is a gripper outside the sphere. When you hold down the left mouse button, the gripper grabs the sphere; when you let go of the button, the gripper releases the sphere. Moving the mouse while holding the button down causes the sphere (and hence the object) to move in the same direction as the mouse.

In addition to the two solids you should also see two wire-frame boxes in the camera window. These are the "bounding boxes" of the two objects. By default Geomview puts a bounding box around each object that it displays so that you have an idea of how large it is.

Notice that as you move the mouse around the tetrahedron and dodecahedron move as a unit. That is because by default what you are actually moving is the "World". To move an individual object instead of the whole world, move the mouse cursor to the *Targets* browser in the *Main* panel. Click (any button) on the word *tetra*. This makes the tetrahedron be the "target object". Now move the cursor back to the camera window and you can rotate just the tetrahedron.

The motion that you have been applying up to now has been rotation, because that is the motion mode that is selected in the *Tools* panel. To translate instead, click on the *Translate* button. Now when you move the mouse in the camera window while holding down the left button, the tetrahedron (which should still be the target object from before) will translate in the direction you move the mouse. Notice that you can translate it beyond the edge of the window as long as you keep holding the left mouse button down. If you lift up on the mouse button while moving the mouse, the tetrahedron will keep going. It moves rather rapidly so it is very easy to lose track of where it is.

If you accidentally lose the tetrahedron by translating it too far out of the view of the window, you can get it back by clicking on the *Center* button in the *Tools* panel. This causes it to come back to its initial position.

Click on the *Center* button to bring the tetrahedron home, and then translate it off to one side so that you can completely see the dodecahedron.

Your world now has two objects in it that are beside each other. You should see the dodecahedron in the middle of the window and maybe part of the tetrahedron off to one side. Go back to the *Targets* browser in the *Main* panel and click on "World" to select the whole world again. Now click on the *Look At* button in the *Tools* panel. You should see the dodecahedron and the tetrahedron in the middle of the window next to each other (see Figure 2.2). The *Look At* button positions the camera in such a way that the target object is centered in the window.

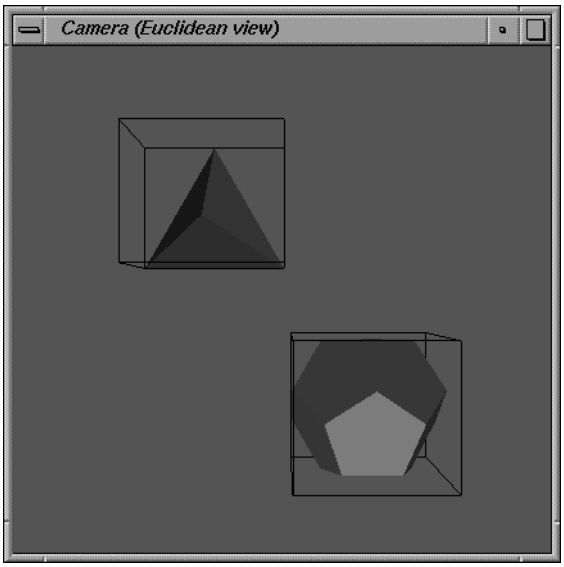


Figure 2.2: Looking at the world.

Now put the cursor over the middle of the dodecahedron and double-click the right mouse button. This means click it down-and-up two times in rapid succession. Notice that the dodecahedron becomes the target object; you can see this in the *Targets* browser in the *Main* panel. Double-clicking the right mouse button on an object is another way to make it the target object.

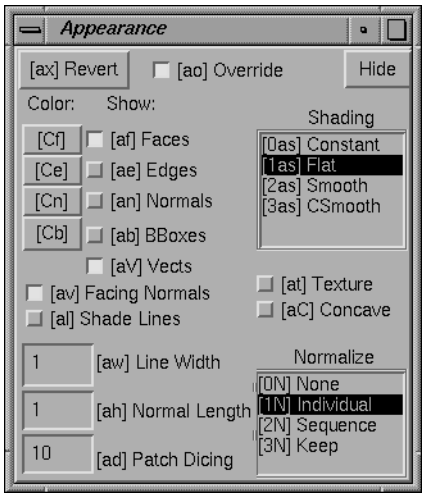


Figure 2.3: The Appearance Panel.

Go to the *Inspect* menu at the top of the *Main* panel and select *Appearance*. This brings up the *Appearance* panel. When it appears, if it partially obscures another Geomview window you can move it off to one side by dragging its frame with the middle mouse button down.

The *Appearance* panel lets you control various things about the way Geomview draws objects. Note the buttons labeled *[af] Faces* and *[ae] Edges*. Click on the *[ae] Edges* one, and notice that Geomview is now drawing the edges of the dodecahedron. Click on it again and the edges go away. Click several times and watch the edges come and go. When you've had enough of this, leave the edges on and click the *[af] Faces* button. This toggles the faces on and off. Click the button again to turn them back on.

Now click on the *[Cf] Faces* button under the word *COLOR*. A color chooser panel should appear (see Figure 2.4).

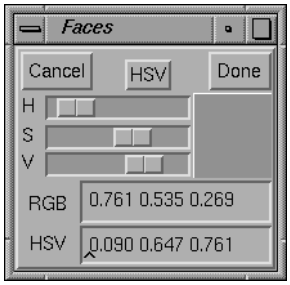


Figure 2.4: Color Chooser Panel.

Note the three sliders, *H*, *S*, and *V*, controlling the color's hue, saturation, and value (lightness). Clicking the *HSV* button gives a different set of sliders, one each for red, green, and blue. Numerical values for both RGB and HSV color systems can be seen or edited at the bottom of the panel. The dodecahedron's previous colors were specified in the file `dodec` that you loaded when we started Geomview. The color that you specify with the color panel overrides the old colors. You can adjust the intensity of the color with the *Intensity* slider. When you find a color that you like, click the *Done* button.

Now put the cursor somewhere over the gray background and double-click the right mouse button; this picks "World" as the target object. Click the *Look At* button to look at the world again.

Notice that in the *Appearance* panel the settings of the buttons have changed from the way you left them with the dodecahedron. That's because the *Appearance* panel always displays the settings for the target object, which is now the world, which still has its default settings.

Click on the *[ab] BBox* button under the word *Draw*. The bounding boxes go away. Now put the cursor back in the camera window. At the keyboard, type the keys `a` b. Notice that the bounding boxes come back. `a` b is the keyboard shortcut for the bounding box toggle button; the string "[ab]" appears on the button to indicate this. Most of Geomview's buttons have keyboard shortcuts that you can use instead if you want. This is useful once you are familiar with Geomview and don't want to have to move around among lots of panels.

Now select the tetrahedron, either by double-clicking the right mouse button on it, or by selecting "tetra" in the *Targets* browser. Then click on the *Delete* button in the *Main* panel. The tetrahedron should disappear. This is how you get rid of an object.

You can also load objects from within Geomview. Click on the *File* menu in the *Main* panel and choose *Open*. The *Files* panel will appear. Below the middle of this panel is a

browser with three lines in it; the second line is a directory with lots of Geomview example files in it. Click on that second line; see Figure 2.5. Scroll down in the list of files until you see `tref.off`. Click on that line, and then click on the *OK* button. A large trefoil-shaped tube will appear in your window. Click the *Hide* button in the *Files* panel to dismiss the panel.

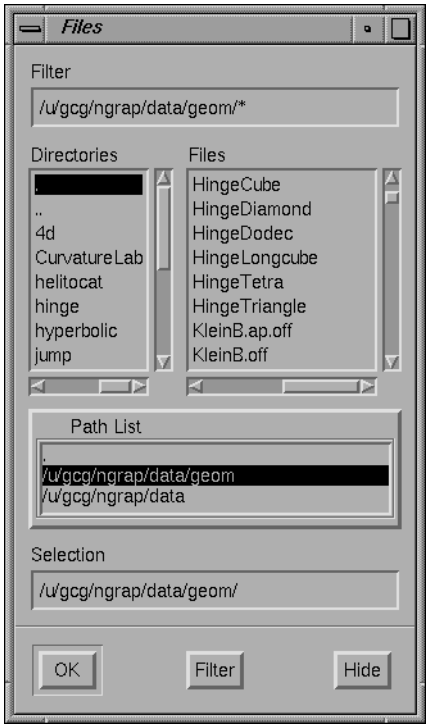


Figure 2.5: The Files Panel.

Now click on the *Reset* button in the *Tools* panel. This causes everything to return to its home position. You should see a dodecahedron and a trefoil knot (see Figure 2.6).

Play around with the trefoil knot and the dodecahedron. Experiment with some of the other buttons in the *Tools* panel. Try coloring the trefoil knot with the *Appearance* panel.

For a tutorial on how to create your own objects to load into Geomview, see file `doc/oogltour` distributed with Geomview. The things in that file will be incorporated into a future version of this manual.

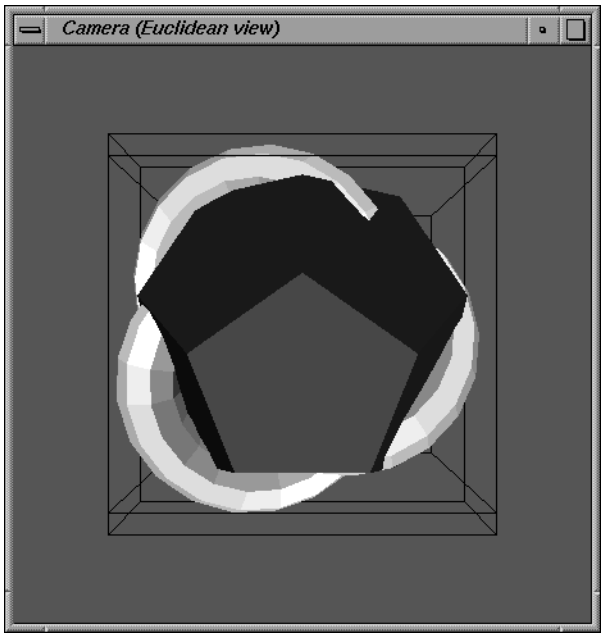


Figure 2.6: Trefoil and Dodecahedron.

3 Interaction

This chapter describes how you interact with Geomview through the mouse and keyboard.

3.1 Starting Geomview

The usual way to start Geomview is to type *geomview* `Enter` in a shell window (`Enter` means hit the "Enter" key). It may take Geomview a few seconds to start up; one or more windows will appear and you can begin interacting with Geomview immediately.

It is also possible to specify actions for Geomview to perform at startup time by giving arguments in the shell command line. See Section 3.2 [Command Line Options], page 23.

3.2 Command Line Options

Here are the command line options that Geomview allows:

'`-b` *r g* b' Set the window background color to the given *r g* b values.

'`-c` *file*' Interpret the GCL commands in *file*, which may be the special symbol `-` for standard input. For a description of GCL, See Chapter 7 [GCL], page 107.

'`-c` *command*'
Commands may also be supplied literally, as in

```
-c "(ui-panel main off)"
```

Since *command* includes parentheses, which have special meaning to the shell, *command* must be quoted. Multiple *-c* options are allowed.

'`-wins` *nwins*'
Causes Geomview to initially display *nwins* camera windows.

'`-wpos` *width*`,`*height*`[@`*xmin*`,`*ymin*`]`'
Specifies the initial location and size of the first camera window. The values for *width*, *height*, *xmin*, and *ymin* are in screen (pixel) coordinates.

'`-M[cg][ps[un|in|in6]]` *PIPENAME*`|`*TCPPORT*'
The '`-M`' option accepts modifiers: a '`g`' suffix expects geometry data (the default), while a '`c`' suffix expects GCL commands. A '`p`' implies the connection should use a named pipe (the default on everything except on the NeXT), while '`s`' implies using a UNIX-domain socket (the default on the NeXT). Since version 1.9 of Geomview Internet domain sockets are also supported; use '`sin`' to make Geomview listen on the IPv4 port given by *TCPPORT*, or use '`sin6`' to make Geomview listen on an IPv6 port (also as specified by *TCPPORT*). '`sun`' is a synonym for '`s`', i.e. use the Unix domain socket with the name *PIPENAME*. If *PIPENAME* starts with a slash ('/'), then it is assumed to be an absolute pathname, otherwise the named pipe or socket is created below `${TMPDIR}/geomview/`.

Listening to command streams on TCP ports can be a security risk, as Geomview itself does not take any security precautions, it simply executes all commands fed to it through the network socket. This also implies that disk-io can be initiated remotely.

Examples:

`-M` *`objectname`*

Display (possibly dynamically changing) geometry sent from the programs `geomstuff` or `togeomview`. This actually listens to the named pipe `/tmp/geomview/objectname`; you can achieve the same effect with the shell commands:

```
mkdir /tmp/geomview
mknod /tmp/geomview/objectname p
```

(assuming the directory and named pipe don't already exist), then executing the GCL command:

```
(geometry objectname  < /tmp/geomview/objectname)
```

(see Section 7.2.57 [geometry], page 116)

`-Mc` *`pipename`*

Like '`-M`' above, but expects GCL commands, rather than OOGL geometry data, on the connection.

`-Mcs fred` Read commands from the UNIX-domain socket named. `/tmp/geomview/fred`

`-Mcsin 40000`

Read commands from the IPv4 port '`40000`'. Geomview itself does not take any security precautions, so this can be a security risk.

'`-noopengl`'

Disable the use of OpenGL for (possibly) hardware accelerated rendering, even if the Geomview binary has support for OpenGL compiled in. This also disables the support for transparency and textures in the camera windows. RenderMan snapshots still will have correct transparency and some limited texture support.

'`-nopanels`'

Start up displaying no panels, only graphics windows. Panels may be invoked later as usual with the *`Px`* keyboard shortcuts or with the `ui-panel` command. See Section 7.2.149 [ui-panel], page 134.

'`-noinit`' Read no initialization files. By default, geomview reads the system-wide `.geomview` file, followed by those in `${HOME}/.geomview` and `./.geomview`.

'`-e` *`module`*'

Start an external module; *module* is the name associated with the module, appearing in the main panel's Applications browser, as defined by the `emodule-define` command. See Section 7.2.40 [emodule-define], page 114.

'`-start` *`module args ...`*'

Like -e but allows you to pass arguments to the external module. "–" signals the end of the argument list; the "–" may be omitted if it would be the last argument on the Geomview command line.

'`-run` *`shell-command args ...`*'

Like -start but takes the pathname of executable of the external module instead of the module's name. The pathnames of all known module directories are appended to the UNIX search path when invoking *shell-command*.

3.3 Basic Interaction: The Main Panel

Normally when you invoke Geomview, three windows appear: the *Main* panel, the *Tools* panel, and one camera window. Geomview has many other windows but most things can be done with these three and so by default the others do not appear. This section of the manual introduces some basic concepts that are used throughout the rest of the manual and describes the *Main* panel.

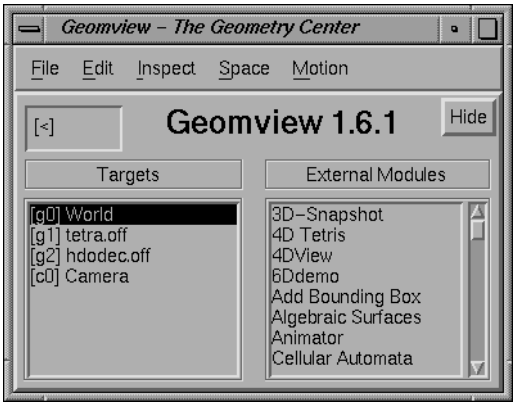


Figure 3.1: The Main Panel

Geomview can display an arbitrary number of objects simultaneously. The *Targets* browser in the *Main* panel displays a list of all the objects that Geomview currently knows about. This browser has a line for each object that you have loaded, plus some lines for other objects. One of the other objects is called `World` and corresponds to the all the currently loaded objects, treated as if they were one object. Most of the operations that you can do to one object, such as applying a motion or changing a color, can also be done to the "World" object.

The *Targets* browser also has an entry for each camera. By default there is only one camera; it is possible to add more of them via the *New Camera* entry of the *Main* panel's *File* menu. Geomview treats cameras in much the same way as it does geometric objects. For example, you can move cameras around and add them and delete them just as with geometric objects. Cameras do not usually show up in the display as an object that you see. Each camera has a separate camera window which displays the view as seen by that camera. (It is possible for each camera to display a geometric representation of other cameras. See Section 3.7 [Cameras], page 39.)

Because Geomview treats cameras and geometric objects very similarly, the term *object* in this documentation is used to refer to either one. When we need to distinguish between the two kinds of objects, we use the term *geom* to denote a geometric object and the word *camera* to denote a camera.

The object which is selected (highlighted) in the *Targets* browser is called the *target* object. This is the object that receives any actions that you do with the mouse or keyboard. You can change the target object by selecting a different line in the *Targets* browser. Another way to change the target object is to put the mouse cursor directly over a geometry in a camera window and rapidly double-click the right mouse button. This process is called *picking*; the picked object becomes the new target.

Geomview objects are all known by two names, both of which are shown in the *Targets* browser. The first name given there, which appears in square brackets ([ ]), is a short name assigned by Geomview when you load the object. It consists of the letter 'g' for geometries and 'c' for cameras, followed by a number. The second name is a longer more descriptive name; by default this is the name of the file that the object was loaded from. The two names are equivalent as far as Geomview is concerned; at any point where you need to specify a name you can give either one.

To manipulate an object, make sure you that the object you want to move is the target object, and put the mouse cursor in a camera window. Motions are applied by holding down either the left or middle mouse button and moving the mouse. There are several different motion "modes", each for applying a different kind of motion. The *MOTION MODE* browser in the *Main* panel indicates the current motion mode. The default is "Rotate". You can change the current motion mode by selecting a new one in the *MOTION MODE* browser, or by using the *Tools* panel. For more information about motion modes, See Section 3.5 [Mouse Motions], page 28.

The *Modules* browser lists Geomview external modules. An external module is a separate program that interacts with Geomview to extend its functionality. For information on external modules, See Chapter 6 [Modules], page 84.

The menu bar at the top of the main panel offers menus for common operations.

To create new windows, load new objects, save objects or other information, or quit from geomview, see the *File* menu.

To copy or delete objects, see the *Edit* menu.

You can invoke any panel from the *Inspect* menu.

The *Space* menu lets you choose whether geomview operates in Euclidean, Hyperbolic, or Spherical mode. Euclidean mode is selected by default. For details about using *Hyperbolic* and *Spherical* spaces, See Chapter 8 [Non-Euclidean Geometry], page 138.

Most actions that you can do through Geomview's panels have equivalent keyboard shortcuts so that you can do the same action by typing a sequence of keys on the keyboard. This is useful for advanced users who are familiar with Geomview's capabilities and want to work quickly without having to have lots of panels cluttering up the screen. Keyboard shortcuts are usually indicated in square brackets ([ ]) near the corresponding item in a panel. For example, the keyboard shortcut for *Rotate* mode is 'r'; this is indicated by "[r]" appearing before the word "Rotate" in the *MOTION MODE* browser. To use this keyboard shortcut, just hit the `r` key while the mouse cursor is in any Geomview window. Do not hit the `Enter` key afterwards.

Some keyboard shortcuts consist of more than one key. In these cases just type the keys one after the other, with no `Enter` afterwards. Keyboard shortcuts are case sensitive.

Many keyboard shortcuts can be preceded by a numeric parameter. For example, typing `ae` toggles the state of drawing edges, while `1ae` always enables edge drawing.

The *keyboard* field in the upper left corner of the *Main* panel echos the current state of keyboard shortcuts.

For a list of all keyboard shortcuts, press the `?` key.

3.4 Loading Objects Into Geomview

There are several ways to load an object into Geomview.

the *Files* panel

If you click the *Load* button in Geomview's *Main* panel, the *Files* panel will appear.



Figure 3.2: The Files Panel.

This panel lets you select a file from a variety of directories. The top of the panel is a standard Motif file browser. Below this is a list of directories on Geomview's standard search path; click on one of these to browse files in that directory.

To select a file, double-click on its name in the browser at upper right, or click on its name and press the `Enter` key, or type the file's name into the text box at the bottom of the browser and press `Enter`.

If the selected file contains OOGL geometric data, it will be added to the geomview *Targets* browser. If it contains GCL commands instead, the file will be interpreted. See Chapter 7 [GCL], page 107.

When the *Files* panel first appears, the directory selected in the directory browser is the current directory — the one from which you invoked Geomview. The file browser shows *all* the files in this directory, including ones that are not Geomview files. If you try to load a file that doesn't contain either an OOGL object or Geomview commands, Geomview will print out an error message.

The directory browser also lists a second and third directory in addition to the current directory. The second one, which ends in `data/geom`, is the Geomview

example data directory. This contains a wide variety of sample objects. It also contains several subdirectories. In particular, the `hyperbolic` and `spherical` subdirectories have sample hyperbolic and spherical objects, respectively. Directory entries in the browser look just like file entries; to view a subdirectory, click on it.

The third directory shown in the directory browser, which ends in `geom`, contains several subdirectories with other Geomview files in them. These are used less frequently than the ones in the `data/geom` directory.

You can change the list of directories shown the *Files* panel's directory browser by using the `set-load-path` command; see Section 7.2.124 [set-load-path], page 128.

the < keyboard shortcut:

If you type < in any Geomview window, the *Load* panel will appear. This is a small version of the *Files* panel; it contains a text field in which you can enter the name of a file to load (or a GCL command surrounded by parentheses). After typing the name of the file to load, type `Enter`; Geomview will load the file as if you had loaded it with the *Add* button in the *Files* panel. If, after bringing up the small load panel with <, you decide you want to use the larger *Files* panel after all, press the *File Browser* button.

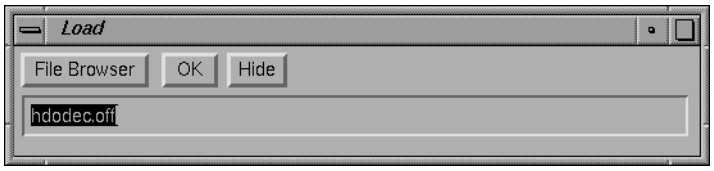


Figure 3.3: The Load Panel.

geometry loading commands:

The `load`, `geometry`, `new-geometry`, and `read` GCL commands allow you to load an object into Geomview; See Chapter 7 [GCL], page 107. See Section 7.2.71 [load], page 119. See Section 7.2.93 [new-geometry], page 123. See Section 7.2.111 [read], page 126.

3.5 Using the Mouse to Manipulate Objects

Geomview lets you manipulate objects with the mouse. There are six different mouse motion modes: *Rotate*, *Translate*, *Cam Fly*, *Cam Zoom*, *Geom Scale*, and *Cam Orbit*. The tools panel has a button for each of these modes; to switch modes, click on the corresponding button. You can also select these through the *Motion Mode* browser on the *Main* panel.

This section describes basic mouse interaction. For details, see Section 3.9 [Commands], page 44.

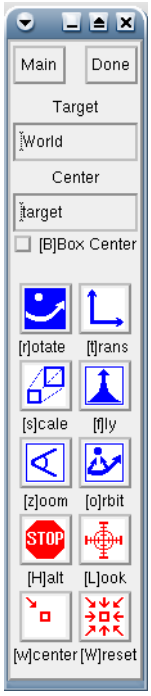


Figure 3.4: The Tools Panel.

Each of the motion modes uses a common paradigm for how the motion is applied. In particular, each depends on the current *target* object and the current *center* object. These are explained in the following paragraphs.

The current target object is shown in the *Target* field in the *Tools* panel. This is the same as the selected object in the *Targets* browser in the *Main* panel, and you can change it by either selecting a new object in the browser, by typing a new entry in the field, or by picking an object in a camera window by double-clicking the right mouse button with the cursor over the object.

The current center object is shown in the *Center* field in the *Tools* panel. Its default value is the special word `"target"`, which means that the center object is whatever the target object is. You can change the center to any object by typing it in the *Center* field. The origin of the center object is held fixed in *Rotate* and *Orbit* modes. Normally the center object is one of the existing geoms listed in the *Targets* browser, and the actual center of rotations is the origin of that object's coordinate system. It is possible, however, to select an arbitrary point of interest on an object as the center. For details, see Section 3.5.1 [Point of Interest], page 32.

It is also possible to toggle the button *BBox Center* to set the center of motion to the center of the current object's bounding box. Once toggled, the active geometry's bounding box center will become the center of motion. If you select another object, then the center of motion will become the center of that object's bounding box. Nothing changes when a camera or the *World* is selected; you have to type in the word `target` in the *Center* field to reset to the default.

You apply a mouse motion by holding down either the left or middle mouse button with the cursor in a camera window and moving the mouse. Most of the modes have *inertia*, which means that if you let go of the button while moving the mouse, the motion will continue. It may be helpful to imagine the mouse cursor as being a gripper; when you hold a mouse button down, it grips the target object and you can move it. When you let go of the mouse button, the gripper releases the object. Letting go of the mouse button while moving the mouse is like throwing the object — the object continues moving independent of the mouse. Inertia can be turned off; see the *Main* panel's *Motion* menu, described below.

Generally, the left mouse button controls motion in the screen plane, while the middle mouse controls motion along or around the forward direction.

Pressing the shift key while dragging with left or middle mouse buttons in most motion modes gives slow-speed motions, useful for fine adjustment.

You can pick any point on an object (not just its origin) as the center of motion by holding down the shift key while clicking the right mouse button; this chooses a point of interest.

Rotate In *Rotate* mode, hold the left mouse button down to rotate the target object about the center object. Rotation proceeds in the direction that you move the mouse. Specifically, the axis of rotation passes through the origin of the center object, is parallel to the camera view plane, and is perpendicular to the direction of motion of the mouse. When the center is "target", this means that the target object rotates about its own origin.

The middle mouse button in *Rotate* mode rotates the target object about an axis perpendicular to the view plane.

Translate In *Translate* mode, hold the left mouse button down to translate the target object in the direction of mouse motion. The middle mouse button translates the target along an axis perpendicular to the view plane.

In Euclidean space, the center object is essentially irrelevant for translations. In hyperbolic and spherical spaces, where translations have a unique axis, this axis is chosen to go through the origin of the center object.

Cam Fly *Cam Fly* is a crude flight simulator that lets you fly around the scene. It works by moving the camera. Move the mouse while holding the left mouse button down to point the camera in a different direction. To move forward or backward, hold down the middle button and move the mouse vertically. Both of these motions have inertia; typically the easiest way to fly around a scene is to give the camera a slight forward push by letting go of the middle button while moving the mouse upward, and then using the left button to steer.

Cam Fly affects the camera window that the mouse is in; it ignores the target object and the center object.

Cam Orbit

Cam Orbit mode lets you rotate the current camera around the current center. The left mouse button does this rotation. The middle mouse button in *Cam Orbit* mode acts as in *Cam Fly* mode: it moves the camera forward or backward.

In general *Cam Orbit* does not move the target object, although if the current camera is selected as the target and the center is also the target, it will pivot that camera about itself just as in *Cam Fly* mode.

Cam Zoom

Cam Zoom mode lets you change the current camera's field of view with the mouse; hold the left mouse button down and move the mouse to change it. The numeric value of the field of view is shown in the *FOV* field in the *Camera* panel.

Geom Scale

Geom Scale mode lets you enlarge or shrink a geom. It operates on the target object if that object is a geom. If the target is a camera, *Geom Scale* operates on the geom that was most recently the target object. Moving the mouse while holding down the left mouse button scales the object either up or down, depending on the direction of mouse motion. The center of the applied scaling transformation is the center object.

Scaling is meaningful only in Euclidean space; attempts to scale are ignored in other spaces.

Geom Scale mode does not have inertia.

The *Stop*, *Look At*, *Center*, and *Reset* buttons on the *Tools* panel perform actions related to motions but do not change the current motion mode.

Stop The *Stop* button causes all motions to stop. It affects all moving objects, not just the target object. Its keyboard shortcut is `H`.

The keyboard command `h`, which does not correspond to a panel button, stops the current motion for the target object only.

Look At The *Look At* button causes the current camera to be moved to a position such that it is looking at the target object, and such that the target object more or less fills the window.

The Look At command is unreliable in non-Euclidean spaces.

Center The *Center* button undoes the target object's transformation, moving it back to its home position, which is where it was when you originally loaded it into Geomview.

Reset The *Reset* button stops all motion and causes all objects to move back to their home positions.

The *Tools* panel also sports a *Main* button, to invoke the main panel in case it was dismissed or buried, and a *Done* button to close the *Tools* panel.

The *Main* panel's *Motion* menu has special controls affecting how mouse motions are interpreted; the toggles are also accessible through a GCL command. See Section 7.2.148 [ui-motion], page 133.

[ui] Inertia

Normally, moving objects have inertia: if the mouse is still moving when the button is released, the selected object continues to move. When *Inertia* is off, objects cease to move as soon as you release the mouse.

[uc] Constrain Motion

It's sometimes handy to move an object in a direction aligned with a coordinate axis: exactly horizontally or vertically. Selecting *Constrain Motion* changes the interpretation of mouse motions to allow this; approximately-horizontal or approximately-vertical mouse dragging becomes exactly horizontal or vertical motion. Note that the motion is still along the X or Y axes of the camera in which you move the mouse, not necessarily the object's own coordinate system.

[uo] Own Coordinates

It's sometimes handy to move objects with respect to the coordinate system where they were defined, rather than with respect to some camera's view. While *Own Coordinates* is selected, all motions are interpreted that way: dragging the mouse rightward in translate mode moves the object in its own +X direction, and so on. May be especially useful in conjunction with the *Constrain Motion* button.

3.5.1 Selecting a Point of Interest

It is sometimes useful to specify a particular point on some object in a geomview window as the center point for mouse motions. You can do this by shift-clicking the right mouse button (i.e. click it once while holding down the shift key on the keyboard) with the cursor over the desired point. This point then becomes the *point of interest*. The point of interest must be on an existing object.

Selecting a point of interest simplifies examining a small portion of a larger object. Shift-right-click on an interesting point, and select *Orbit* mode. Use the middle mouse button to approach, and the left mouse to orbit the point, examining the region from different directions.

When you have selected a point of interest, the current center object changes to an object named "CENTER", which is an invisible object located at the point of interest. In addition, mouse motions for the window in which you made the selection are adjusted so that the point of interest follows the mouse.

You can change the point of interest at any time by selecting a new one by shift-clicking the right mouse button again. You can cancel the point of interest altogether by shift-clicking the right mouse button with the cursor on the background (i.e. not on any object). This changes the center object back to its default value, "target".

The object named "CENTER", which serves as the center object for the point of interest, is a special kind of geom called an "alien". It does not appear in the *Targets* browser. By default it has no geometry associated with it and hence is invisible. You can, however, explicitly give it some geometry using a GCL command, causing it to appear. Use the `geometry` command for this: `(geometry CENTER` *`geometry`*`)`, where *geometry* is any valid geometry. For example, `(geometry CENTER { < xyz.vect })` causes the file `xyz.vect`, which is one of the standard example files distributed with geomview, to be used at the geometry for CENTER. See Section 7.2.57 [geometry], page 116.

What happens internally when you select a point of interest is that the center is set to the object called CENTER, and that object is positioned at the point of interest. In addition, in order for mouse motions to track the point of interest, the current camera's

focal length is set to be the distance from the camera to the point of interest. You can accomplish this via GCL with the following commands:

```
(if (real-id CENTER) nil (new-alien CENTER {}))
(ui-center CENTER)
(transform-set CENTER universe universe translate x y z)
(merge camera cam-id { focus d })
```

where `(x,y,z)` are the (universe) coordinates of the point of interest, and *d* is the distance from that point to the current camera, *cam-id*. The first command above creates the "alien" CENTER if it does not yet exist.

3.6 Changing the Way Things Look

Geomview uses a hierarchy of appearances to control the way things look. An *appearance* is a specification of information about how something should be drawn. This can include many things such things as color, lighting, material properties, and more. Appearances work in a hierarchal manner: if a certain appearance property, for example face color, is not specified in a particular object's appearance, that object is drawn using that property from the parent appearance. If both the parent and the child appearance specify a property, the child's setting takes precedence unless the parent appearance is set to override.

Every geom in Geomview has an appearance associated with it. There is also an appearance associated with the "World" geom, which serves as the parent of each individual geom's appearance. Finally, there is a global "base" appearance, which is the parent of the World appearance.

The base appearance specifies reasonable values for all appearance information, and by default none of the other appearances specify anything, which means they inherit their values from the base appearance. This means that by default all objects are drawn using the base appearance.

If you change a certain appearance property for a geom, that property is used in drawing that geom. The parent appearance is used for any properties that you do not explicitly set.

Geomview has three panels which let you modify appearances.

3.6.1 The Appearance Panel

The *Appearance* panel lets you change most common appearance properties of the target object.

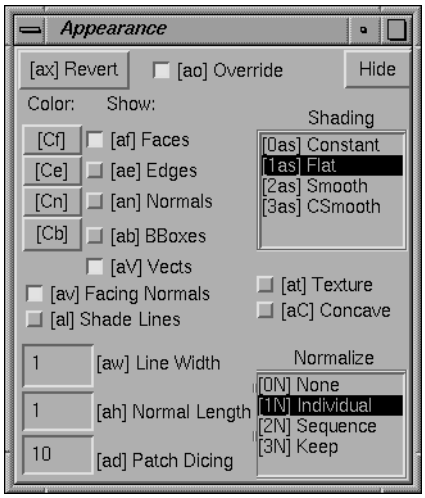


Figure 3.5: The Appearance Panel.

If the target is an individual geom, then changes you make in the appearance panel apply to that geometry's appearance. If the target is the World, then appearance panel changes apply to the World appearance *and* to all individual geom appearances. (Users have found that this is more desirable than having the changes only apply to the World appearance.) If the target is a camera, then appearance panel changes apply to the geom that was most recently the target.

The five buttons near the upper left corner under the word *Show* control what parts of the target geom are drawn.

Faces This button specifies whether faces are drawn.

Edges This button specifies whether edges are drawn.

BBox This button specifies whether the bounding box is drawn.

Vects This button specifies whether VECT objects are drawn. VECTs are a type of OOGL object that represent points and line segments in 3-space; they are distinct from edges of other kinds of objects, and it is sometimes desirable to have separate control over whether they are drawn.

Normals This button specifies whether surface normal vectors are drawn.

The four buttons under *Color* labeled *Faces*, *Edges*, *Normals*, and *BBox* let you specify the color of the corresponding aspect of the target geom. Clicking on one of them brings up a color chooser panel.

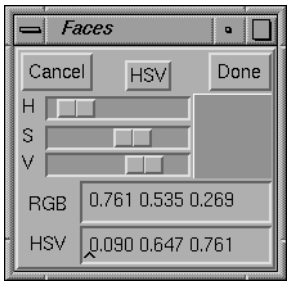


Figure 3.6: Color Chooser Panel.

This panel offers two sets of sliders: H(ue) S(aturation) V(alue), or R(ed) G(reen) B(lue), each in the range 0 through 1. The square shows the current color, which is given numerically in both HSV and RGB systems in the corresponding text boxes.

In the HSV color system, hue H runs from red at 0, green at .333, blue at .667, and back to red at 1.0. Saturation gives the fraction of white mixed into the color, from 0 for pure gray to 1 for pure color. Value gives the brightness, from 0 for black to 1 for full brightness.

Pressing the *RGB* or *HSV* button at top center switches the sliders to the other color system. You can adjust colors either via the sliders, or by typing in either the RGB or HSV text boxes.

Click *OK* to accept the color that you have chosen, or *Cancel* to retain the previous color setting.

The *SHADING* browser lets you specify the shading model that Geomview uses to paint the target geom.

`Constant` Every face of the object is drawn with a constant color which does not depend on the location of the face, the camera, or the light sources. If the object does not contain per-face or per-vertex colors, the diffuse color of the object's appearance is used. If the object contains per-face colors, they are used. If the object contains per-vertex colors, each face is painted using the color of its first vertex.

`Flat` Each face of the object is drawn with a color that depends on the relative location of the face, the camera, and the light sources. The color is constant across the face but may change as the face, camera, or lights move.

`Smooth` Each face of the object is drawn with smoothly interpolated colors based on the normal vectors at each vertex. If the object does not contain per-vertex normals, this has the same effect as flat shading. If the object has reasonable per-vertex normals, the effect is to smooth over the edges between the faces.

`CSmooth` Each face of the object is drawn with exactly the specified color(s), independent of lighting, orientation, and material properties. If the object is defined with per-vertex colors, the colors are interpolated smoothly across the face; otherwise the effect is the same as in Constant shading style.

`VCflat` A combination of `CSmooth` and `Flat` shading. So the shading is constant on each face, depending on the relative orientation of the light sources, the camera and the surface normal of the face.

The *Facing Normals* button on the *Appearance* panel indicates whether or not Geomview should arrange that normal vectors always face the viewer. If a normal vector points away from the viewer the color of the corresponding face or vertex usually is darker than is desired. Geomview can avoid this by using the opposite normal in shading calculations. This is the default. Using *Facing Normals* can give strange flickering dark or light shading effects, though, near the horizon of a fairly smooth facetted object. Press this button to use the normals given with the object.

The three text fields in the lower left corner of the *Appearance* panel are:

Line Width
: The width, in pixels, for lines drawn by Geomview.

Normal Length
: This is actually a scale factor; when normal vectors are drawn, Geomview draws them to have a length that is their natural length times this number.

Patch Dicing
: Geomview draws Bezier patches by first converting them to meshes. This parameter specifies the resolution of the mesh: if *Patch Dicing* is n, then an n by n mesh is used to draw each Bezier patch. if *Patch Dicing* is 1, the resolution reverts to a built-in default value.

The *Revert* button on the *Appearance* panel undoes all settings in the target appearance. This causes the target geom to inherit all its appearance properties from its parent.

The *Appearance* panel's *Override* button determines whether appearance controls should override settings in the objects themselves – for example, setting the face color will affect all faces of objects with multi-colored facets. Otherwise, appearance controls only provide settings which the objects themselves do not specify. By default, *Override* is enabled. This button applies to all objects, and to all appearance-related panels.

Normalization is a kind of scaling; Geomview can scale an object so that it fits within a certain region. The main point of normalization is to allow you to easily view all of an object without having to worry about how big it is. We are gradually replacing Geomview's normalization feature with more robust camera positioning features. In general, the best way to make sure you are seeing all of an object is to use the *Look At* button on the *Tools* panel. Normalization may be completely replaced by this and other features in a future version of Geomview.

Normalization is a property that applies to each geom separately. The *NORMALIZE GEOMETRY* browser affects the normalization property of target geom. If the target geom is "World", it affects all geoms.

None Do no normalization.

Individual Normalize this geom to fit within a unit sphere.

Sequence This resembles "Individual", except when an object is changing. Then, "Individual" tightly fits the bounding box around the object whenever it changes and normalizes accordingly, while "Sequence" normalizes the union of all variants of the object and normalizes accordingly.

Keep This leaves the current normalization transform unchanged when the object changes. It may be useful to apply "Individual" or "Sequence" normalization

to the first version of a changing object to bring it in view, then switch to "Keep".

3.6.2 The Materials Panel

The *Materials* panel controls material properties of surfaces. It works with the target object in the same way that the *Appearance* panel does.

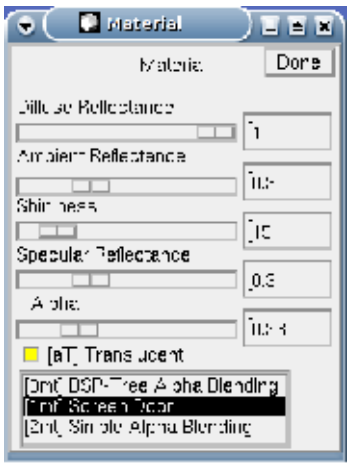


Figure 3.7: The Materials Panel.

Translucent

This button determines whether translucency is enabled. Geomview supports three different flavours of translucency:

Alpha-blending with BSP-tree depth-sorting

This is the most accurate flavour for online viewing, but consumes vast amounts of computation time and memory. In this mode single translucent objects are displayed correctly with respect to themselves; however multiple translucent objects may appear in the wrong order on the screen. The notion *object* means here: top-level geometries as displayed in the geometry target browser.

Screen Door Translucency

If the machine support OpenGL then there is support for kind-of translucency by masking out (completely) transparent pixels by means of a stipple mask. This is currently very experimental, and the result is somewhat ugly, but fast.

Alphe-blending without depth-sorting

This is the old naive way of doing quick and dirty translucency. It is fast, but the results are completely incorrect.

When transparency is enabled, a RenderMan snapshot will contain the alpha information, a RenderMan compliant renderer can then generate high-quality pictures, including correct translucency.

Alpha This slider determines the opacity/transparency when transparency is enabled. 0 means totally transparent, 1 means totally opaque.

Diffuse Reflectance
: This slider controls the diffuse reflectance of a surface. This has to do with how much the surface scatters light that it reflects.

Shininess
: This slider controls how shiny a surface is. This determines the size of specular highlights on the surface. Lower values give the surface a duller appearance.

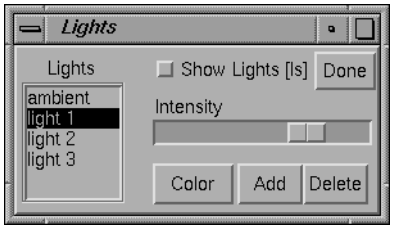
Ambient Reflectance
: This slider controls how much of the ambient light a surface reflects.

Specular Reflectance
: This slider controls the specular reflectance of a surface. This has to do with how directly the surface reflects light rays. Higher values give brighter specular highlights.

Done
: This button dismisses the *Materials* panel.

3.6.3 The Lighting Panel

The *Lights* panel controls the number, position, and color of the light sources used in shading.



The Lighting Panel.

The *Lighting* panel is different from the *Appearance* and *Material* panels in that it always works with the base appearance. This is because it usually makes sense to use the same set of lights for drawing all objects in your scene.

LIGHTS
: The *LIGHTS* browser shows the currently selected light. Changes made using the other widgets on this panel apply to this light. There is always at least one light, the ambient light.

Intensity
: This slider controls the intensity of the current light.

Color
: This button brings up a color chooser which lets you select the color of the current light.

Add
: This button adds a light.

Delete
: This button deletes the current light.

Show Lights
: This button lets you see and change the positions of the light sources in a camera window. Each light is drawn as long cylinder which is supposed to remind you of a light beam. When you click on the *Show Lights* button Geomview goes into "light edit" mode, during which you can rotate current light by holding down the left mouse button and moving the mouse. Lights placed in this way

are infinitely distant, so what you are changing is the angular position. Click on the *Show Lights* button again to return to the previous motion mode and to quit drawing the light beams.

Done This button dismisses the *Lighting* panel.

Geomview's *Appearance*, *Materials*, and *Lighting* panels are constructed to allow you to easily do most of the appearance related things that you might want to do. The appearance hierarchy that Geomview supports internally, however, is very complex and there are certain operations that you cannot do with the panels. The Geomview command language (GCL) provides complete support for appearance operations. In particular, the `merge-baseap` command can be used to change the base appearance (which, except for lighting, cannot be changed by Geomview's panels). The `merge-ap` command can be used to change an individual geom's appearance. Appearances can also be specified in OOGL files; for details, see Section 4.1.10 [Appearances], page 53. See Section 7.2.81 [merge-baseap], page 121. See Section 7.2.79 [merge-ap], page 121.

3.7 Cameras

A camera in Geomview is the object that corresponds to a camera window. By default there is only one camera, but it is possible to have as many as you want. You can control certain aspects of the way the world is drawn in each camera window via the *Cameras* panel.

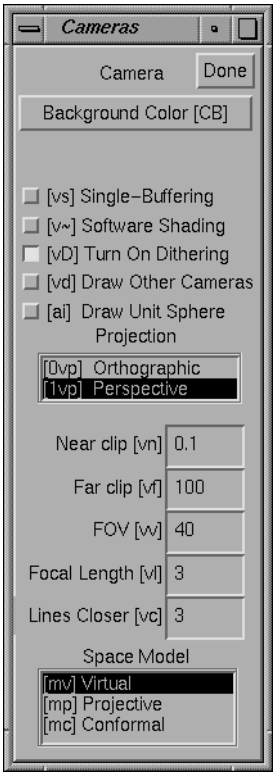


Figure 3.8: The Cameras Panel.

If the target object is a camera, the *Cameras* panel affects that camera. If the target object is not a camera, the *Cameras* panel affects the *current camera*. The current camera is the camera of the window that the mouse cursor is in, or was in most recently if the cursor is not in a camera window. Thus, if you use the keyboard shortcuts for the actions in the *Cameras* panel while the cursor is in a camera window, the actions apply to that camera, unless you have explicitly selected another camera.

To create new camera windows, use the `v+` keyboard shortcut, or see the *File* menu on the *Main* panel.

Single-Buffering
: Normally, geomview windows are *double-buffered*: geomview draws the next picture into a hidden window, then switches buffers to make it visible all at once. On many systems, the memory for the hidden buffer comes from stealing half the bits in each screen pixel, reducing the color resolution. When single-buffering is enabled, the window flickers as each scene is being drawn, but you may get smoother images with reduced grainy dithering artifacts. Single-buffering is possible if Geomview is compiled with GL or OpenGL, but not with plain-X graphics.

Dither
: Many displays offer less than the 24 bits per pixel (8 bits each of red, green, and blue) conventionally needed to show smooth gradations of color. When trying to show a color not accurately available on the display, Geomview normally *dithers*, choosing pixel colors sometimes brighter, sometimes darker than the desired value, so that the average color over an area is a better approximation to the true color than a single pixel could be. Effectively this loses spatial resolution to gain color resolution. This isn't always desirable, though. Turning *Dither* off gives less grainy, but less accurately colored, images.

Software Shading
: This button controls whether Geomview does shading calculations in software. The default is to let the hardware handle them, and in Euclidean space this is almost certainly best because it is faster. In hyperbolic and spherical space, however, the shading calculations that the hardware does are incorrect. Click this button to turn on correct but slower software shading.

Background Color
: This button brings up a color chooser which you can use to set the background color of the camera's window.

PROJECTION
: This browser lets you pick between perspective and orthogonal projection for this camera.

Near clip
: This determines the distance in world coordinates of the near clipping plane from the eye point. It must be a positive number.

Far clip
: This determines the distance in world coordinates of the far clipping plane from the eye point. It must be a positive number and in general should be larger than the *Near clip* value.

FOV
: This is the camera's field of view, measured in its shorter direction. In perspective mode, it is an angle in degrees. In orthographic mode, it is the linear

size of the field of view. This number can be modified with the mouse in *Cam Zoom* mode.

Focal Length
: The focal length is intended to suggest the distance from the camera to an imaginary plane of interest. Its value is used when switching between orthographic and perspective views (and during stereo viewing), so as to preserve apparent size of objects lying at the focal distance from the camera. Focal length also affects interpretation of mouse-based translational motions. Speed of forward motion (in translate, fly and orbit modes) is proportional to focal length; and objects lying at the focal distance from the camera translate laterally at the same rate as the mouse cursor. Finally, in N-D projection mode, cameras are displaced back by the focal distance from the 3-D projection of the world origin.

Lines Closer
: This number has to do with the way lines are drawn. Normally Geomview's z-buffering algorithm can get confused when drawing lines that lie exactly on surfaces (such as the edges of an object); due to machine round-off error, sometimes the lines appear to be in front of the surface and sometimes they appear behind it. The *Lines Closer* value is a fudge factor — Geomview nudges all the lines that it draws closer to the camera by this amount. The number should be a small integer; try 5 or 10. 0 turns this feature off completely. Choosing too large a value will make lines visible even though they should be hidden.

SPACE MODEL
: This determines the model used to draw the world. It is most useful in hyperbolic and spherical spaces. You probably don't need to touch this browser if you stay in Euclidean space. For more information about these models, see Chapter 8 [Non-Euclidean Geometry], page 138.

  Virtual This is the default model and represents the natural view from inside the space.

  Projective The projective model of hyperbolic and spherical space. Geoms move under isometries of the space, and cameras move by Euclidean motions. By default in the projective model, the Euclidean unit sphere is drawn. In hyperbolic space this is the sphere at infinity. In Euclidean space the projective model is the same as the virtual model except that the sphere is drawn by default.

  Conformal
  : The conformal model of hyperbolic and spherical space. Geoms move under isometries of the space, and cameras move by Euclidean motions. In Euclidean space, the conformal model amounts to inverting everything in the unit sphere.

Draw Sphere
: This controls whether Geomview draws the unit sphere. By default the unit sphere appears in the projective and conformal models. In hyperbolic space this is the sphere at infinity. In spherical space it is the equatorial sphere.

Done This button dismisses the *Cameras* panel.

3.8 Saving your work

Geomview's *Save* panel lets you store Geomview objects and other information in files that you can read back into Geomview or other programs.

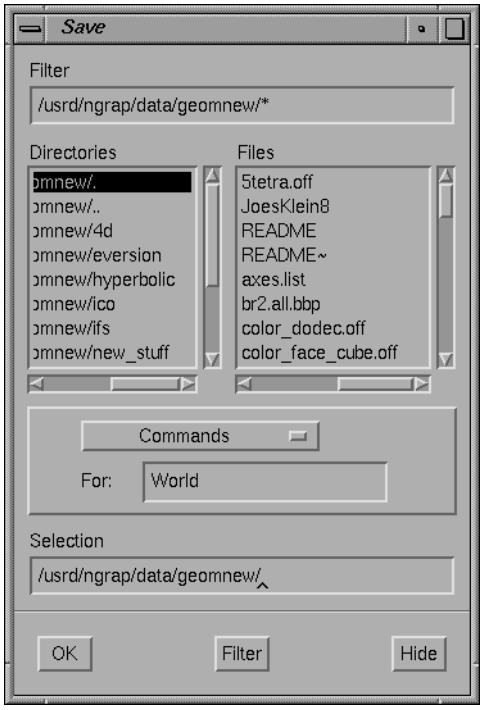


Figure 3.9: The Save Panel.

To use the *Save* panel you select the desired format in the browser next to the word *Save*, enter the name of the object you want to save in the text field next to the word *for*, and enter the name of the file you wish to save to in the long text field next to the word *in*. You can then either hit `Enter` or click on the *OK* button. When the file has been written, the *Save* panel disappears. If you want to dismiss the *Save* panel without writing a file, click the *Cancel* button.

If you specify `-` as the file name, Geomview will write the file to standard output, i.e. in the shell window from which you invoked Geomview.

The possible formats are given below. The kind of object that can be written with each format is given in parentheses.

Commands (any object)
: This write a file of GCL commands containing all information about the object. Loading this file later will restore the object as well as all other information about it, such as appearance, transformations, etc.

Geometry alone (geom)
: This writes an OOGL file containing just the geometry of the object.

Geometry [in world] (geom)
: This writes an OOGL file containing just the geometry of the object, transformed under Geomview's current transformation for this object. Use this if

you have moved the object from its initial position and want to save the new position relative to the world.

Geometry [in universe] (geom)
This writes an OOGL file containing just the geometry of the geom, transformed under both the object's transformation and the world's transformation.

RMan [->tiff] (camera)
Writes a RenderMan file which when rendered creates a tiff image. Transparency and texturing (the latter only to some extent) will be available.

RMan [->frame] (camera)
Writes a RenderMan file which when rendered causes an image to appear in a window on the screen. Transparency and texturing (the latter only to some extent) will be available.

SGI snapshot (camera)
Write an SGI raster file. A bell rings when the snapshot is complete. Only available on SGI systems.

PPM GLX-offscreen snapshot (camera)
Render the complete scene anew into off-screen memory; GLX provides the means to use a Pixmap as rendering area. The advantage of rendering into *off*-screen memory over taking screen snapshot is that the camera windows need not be mapped and even raised at the time the snapshot is taken. So with off-screen snapshot one can safely iconify the camera window (but do not close it!), activate the screen-saver and go to bed while some script advances the scenes and takes snapshots.

PPM Screen snapshot (camera)
Take a snapshot of the given window and save it as a PPM image. If you specify a string beginning with a vertical bar (`|`) as the file name, it's interpreted as a Bourne shell command to which the PPM data should be piped, as in '`| pnmtotiff > snap.tiff`' or '`| convert -geometry 50% ppm:- snap.gif`'.

PPM screen snapshots are only available with GL and Open GL, not plain X graphics. The window should be entirely on the screen. Geomview will ensure that no other windows cover it while the snapshot is taken. It is probably a better idea to use GLX-*off-screen* snapshots, as explained above.

PPM software snapshot (camera)
Writes a snapshot of that window's current view, as a PPM image, to the given file. The file name may be a Bourne shell command preceded by a vertical bar (`|`), as with the PPM screen snapshot. The software snapshot, though, is produced by using a built-in software renderer (related to the X-windows renderer). It doesn't matter whether the window is visible or not, and doesn't depend on GL or OpenGL. It also doesn't support some features, such as texture mapping.

Postscript snapshot (camera)
Writes a Postscript snapshot of the camera's view. It's made by breaking up the scene into lines and polygons, sorting by depth, and generating Postscript

lines and polygons for each one. Advantages over pixel-based snapshot images: resolution is very high, so edges look sharp even on high-resolution printers, or comparable-resolution images are typically much more compact. Disadvantages: depth-sorting gives good results on some scenes, but can be wildly wrong as a hidden-surface removal algorithm for other scenes. Also, Postscript doesn't offer smoothly interpolated shading, only flat shading for each facet.

Camera (camera)
: Writes an OOGL file of a camera.

Transform [to world] (any object)
: Writes an OOGL transform file giving Geomview's transform for the object.

Transform [to universe] (any object)
: Writes an OOGL transform file giving a transform which is the composition of Geomview's transform for the object and the transform for the world.

Window (camera)
: Writes an OOGL window file for a camera.

Panels
: Writes a GCL file containing commands which record the state of all the Geomview panels. Loading this file later will restore the positions of all the panels.

3.9 The Commands Panel

The *Commands* panel lets you type in a GCL command. When you hit `Enter`, Geomview interprets the command and prints any resulting output or error messages on standard output. You can edit the text and hit `Enter` as many times as you like, in general, whenever you hit `Enter` with the cursor in the *Commands* panel, Geomview tries to interpret whatever text you have typed in the text field as a command.

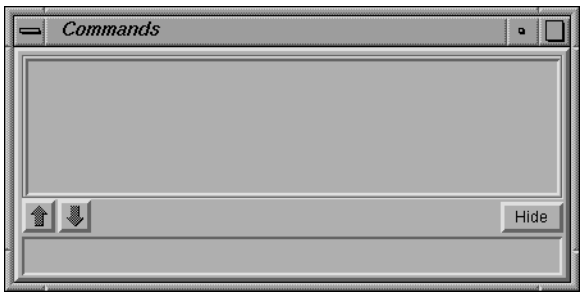


Figure 3.10: The Commands Panel.

3.10 Keyboard Shortcuts

Most actions that you can do through Geomview's panels have equivalent keyboard shortcuts so that you can do the same action by typing a sequence of keys on the keyboard. This is useful for advanced users who are familiar with Geomview's capabilities and want to work quickly without having to have lots of panels cluttering up the screen. Keyboard shortcuts usually are indicated in square brackets ([ ]) near the corresponding item in a panel. For example, the keyboard shortcut for *Rotate* mode is 'r'; this is indicated by `"[r]"` appearing

before the word "Rotate" in the *MOTION MODE* browser. To use this keyboard shortcut just hit the *r* key while the mouse cursor is in any Geomview window. You don't need to press the Enter or SPACE keys.

Some keyboard shortcuts consist of more than one key. In these cases just type the keys one after the other, with no Enter afterwards. Keyboard shortcuts are case sensitive. You can cancel a multi-key keyboard shortcut that you have started by typing any invalid key, for example the space bar.

Keyboard commands apply while the cursor is in any camera window and most control panels.

Many keyboard shortcuts allow numeric arguments which you type as a prefix to the command key(s). For example, the shortcut for *Near clip* in the camera panel is *v n*. To set the near clip plane to '0.5', type *0.5vn*. Commands that don't take a numeric prefix toggle or reset the current value.

Most commands allow one of the following selection prefixes. If none is provided the command applies to the target object.

g	world geom
g#	#'th geom
*g**	All geoms
c	current camera
c#	#'th camera
*c**	All cameras

For example, *g4af* means toggle the face drawing of object *g4*.

Simply typing a selection prefix, like *g4*, doesn't yet select an object; that only happens when a command, like *ae*, follows the prefix. To select an object as the target without doing anything else to it, use the *p* command. So *g3p* selects object g3.

The text field in the upper left corner of the *Main* panel shows the state of the current keyboard shortcut.

In addition to the keyboard shortcuts for the panel commands, there is also a shortcut for picking a target object: type the short name of the object followed by *p*. For example, to select object *g3*, type *g 3 p*. This only works with the short names — the ones that appear in square brackets ([ ]) in the *Targets* browser of the *Main* panel.

Below is a summary of all keyboard shortcuts.

Draw

af	Faces
ae	Edges
an	Normals
ab	Bounding Boxes
aV	Vectors

Shading

`0as`	Constant
`1as`	Flat
`2as`	Smooth
`3as`	Smooth, non-lighted
`aT`	allow transparency
`at`	texture mapping

Other

`av`	eVert normals: always face viewer
`#aw`	Line Width (pixels)
`aC`	handle concave polygons
`#vc`	edges Closer than faces (try 5-100)

Color

`Cf`	faces
`Ce`	edges
`Cn`	normals
`Cb`	bounding boxes
`CB`	background

Motions

`r`	rotate
`t`	translate
`z`	zoom FOV
`f`	fly
`o`	orbit
`s`	scale
`w`	recenter target
`W`	recenter all
`h`	halt
`H`	halt all
`@`	select center of motion (e.g. `g 3 @`)
`L`	Look At object

Viewing

`0vp`	Orthographic view

1vp	Perspective view
vd	Draw other views' cameras
#vv	field of View
#vn	near clip distance
#vf	far clip distance
v+	add new camera
vx	cursor on/off
vb	backfacing poly cull on/off
#vl	focal length
v~	Software shading on/off

Panels

Pm	Main
Pa	Appearance
Pl	Lighting
Po	Obscure
Pt	Tools
Pc	Cameras
PC	Commands
Pf	Files
Ps	Save
P-	read commands from tty
PA	Credits ("about")

Lights

ls	show lights
le	edit lights

Space

me	Euclidean
mh	Hyperbolic
ms	Spherical

Model

mv	Virtual
mp	Projective
mc	Conformal

Other

0N	normalization: none
1N	normalization: each
2N *all*	normalization: all
ui	motion: Inertia
uc	motion: Constrain to axis
uo	motion: object's Own coordinates
<	
Pf	load geometry/command file
dd	delete target object
>	
Ps	save state to file
TV	NTSC mode toggle
p	pick as target object (e.g. *g* *3* *p*) With no prefix, selects the object under the mouse cursor (like double-clicking the right mouse)

4 OOGL File Formats

The objects that you can load into Geomview are called OOGL objects. OOGL stands for "Object Oriented Graphics Library"; it is the library upon which Geomview is built.

There are many different kinds of OOGL objects. This chapter gives syntactic descriptions of file formats for OOGL objects.

Examples of most file types live in Geomview's `data/geom` directory.

4.1 Conventions

4.1.1 Syntax Common to All OOGL File Formats

Most OOGL object file formats are free-format ASCII — any amount of white space (blanks, tabs, newlines) may appear between tokens (numbers, key words, etc.). Line breaks are almost always insignificant, with a couple of exceptions as noted. Comments begin with # and continue to the end of the line; they're allowed anywhere a newline is.

Binary formats are also defined for several objects; See Section 4.1.8 [Binary format], page 51, and the individual object descriptions.

Typical OOGL objects begin with a key word designating object type, possibly with modifiers indicating presence of color information etc. In some formats the key word is optional, for compatibility with file formats defined elsewhere. Object type is then determined by guessing from the file suffix (if any) or from the data itself.

Key words are case sensitive. Some have optional prefix letters indicating presence of color or other data; in this case the order of prefixes is significant, e.g. `CNMESH` is meaningful but `NCMESH` is invalid.

4.1.2 File Names

When OOGL objects are read from disk files, the OOGL library uses the file suffix to guess at the file type.

If the suffix is unrecognized, or if no suffix is available (e.g. for an object being read from a pipe, or embedded in another OOGL object), all known types of objects are tried in turn until one accepts the data as valid.

4.1.3 Vertices

Several objects share a common style of representing vertices with optional per-vertex surface-normal and color. All vertices within an object have the same format, specified by the header key word.

All data for a vertex is grouped together (as opposed to e.g. giving coordinates for all vertices, then colors for all vertices, and so on).

The syntax is

'*x y z*' (3-D floating-point vertex coordinates) or

'*x y z w*' (4-D floating-point vertex coordinates)

optionally followed by

'*nx ny nz*' (normalized 3-D surface-normal if present)

optionally followed by

‘*r g* b a’ (4-component floating-point color if present, each component in range 0..1. The *a* (alpha) component represents opacity: 0 transparent, 1 opaque.)

optionally followed by

‘*s t*’

‘or’

‘*s t u*’

(two or three texture-coordinate values).

Values are separated by white space, and line breaks are immaterial.

Letters in the object's header key word must appear in a specific order; that's the reverse of the order in which the data is given for each vertex. So a ‘CN4OFF’ object's vertices contain first the 4-component space position, then the 3-component normal, finally the 4-component color. You can't change the data order by changing the header key word; an ‘NCOFF’ is just not recognized.

4.1.4 *N*-dimensional Vertices

Several objects share a common style of representing vertices with optional per-vertex surface-normal and color. All vertices within an object have the same format, specified by the header key word.

All data for a vertex is grouped together (as opposed to e.g. giving coordinates for all vertices, then colors for all vertices, and so on).

The syntax for N-dimensional vertices ($N > 3$) is

‘*x[1] x[2] x[3] x[4] ...*’
(N floating-point vertex coordinates) or

‘*x[0] x[1] x[2] x[3] x[4] ...*’
(*(N+1)* floating-point vertex coordinates, if the 4 modifier has been specified in the object's header line)

Note, however, that N-dimensional objects internally always have *(N+1)*-dimensional points; the first component *x[0]* – if present in the object file – is used as homogeneous divisor. This is different from the ordinary 3D case where the 4 modifier generates a 4D object where the homogeneous component implicitly is set to 1.

Color components usually can be specified like for 3D vertices, see Section 4.1.3 [Vertices], page 49, while specifying normals does not make sense.

4.1.5 Surface normal directions

Geomview uses normal vectors to determine how an object is shaded. The direction of the normal is significant in this calculation.

When normals are supplied with an object, the direction of the normal is determined by the data given.

When normals are not supplied with the object, Geomview computes normal vectors automatically; in this case normals point toward the side from which the vertices appear in counterclockwise order.

On parametric surfaces (Bezier patches), the normal at point P(u,v) is in the direction dP/du cross dP/dv.

4.1.6 Transformation matrices

Some objects incorporate 4x4 real matrices for homogeneous object transformations. These matrices act by multiplication on the right of vectors. Thus, if p is a 4-element row vector representing homogeneous coordinates of a point in the OOGL object, and A is the 4x4 matrix, then the transformed point is p' = p A. This matrix convention is common in computer graphics; it's the transpose of that often used in mathematics, where points are column vectors multiplied on the right of matrices.

Thus for Euclidean transformations, the translation components appear in the fourth row (last four elements) of A. A's last column (4th, 8th, 12th and 16th elements) are typically 0, 0, 0, and 1 respectively.

4.1.7 ND Transformation matrices

In the context of N-dimensional space ($N > 3$) some objects incorporate *(N+1)*x*(N+1)* real matrices for homogeneous object transformations. These matrices act by multiplication on the right of vectors. Thus, if p is an *(N+1)*-element row vector representing homogeneous coordinates of a point in the OOGL object, and A *(N+1)*x*(N+1)* is the matrix, then the transformed point is p' = p A.

Note that (unlike for the 4x4 transformation matrices, see Section 4.1.6 [Transformation matrices], page 51) the homogeneous component is located at index **zero**, so the translation components for Euclidean transformations appear in the **zero**-th row (first *(N+1)* elements). A's first column (at column index zero) is typically 1, 0, . . . , 0.

4.1.8 Binary format

Many OOGL objects accept binary as well as ASCII file formats. These files begin with the usual ASCII token (e.g. `CQUAD`) followed by the word `BINARY`. Binary data begins at the byte following the first newline after `BINARY`. White space and a single comment may intervene, e.g.

```
OFF BINARY      # binary-format "OFF" data follows
```

Binary data comprise 32-bit integers and 32-bit IEEE-format floats, both in big-endian format (i.e., with most significant byte first). This is the native format for 'int's and 'float's on Sun-3's, Sun-4's, and Irises, among others.

Binary data formats resemble the corresponding ASCII formats, with ints and floats in just the places you'd expect. There are some exceptions though, specifically in the `QUAD`, `OFF` and `COMMENT` file formats. Details are given in the individual file format descriptions. See Section 4.2.1 [QUAD], page 59, See Section 4.2.5 [OFF], page 62, and See Section 4.2.14 [COMMENT], page 71.

Binary OOGL objects may be freely mixed in ASCII object streams:

```
LIST
{ = MESH BINARY
... binary data for mesh here ...
}
{ = QUAD
```

```
        1 0 0   0 0 1   0 1 0  0 1 0
}
```

Note that ASCII data resumes immediately following the last byte of binary data.

Naturally, it's impossible to embed comments inside a binary-format OOGL object, though comments may appear in the header before the beginning of binary data.

4.1.9 Embedded objects and external-object references

Some object types (`LIST`, `INST`) allow references to other OOGL objects, which may appear literally in the data stream, be loaded from named disk files, or be communicated from elsewhere via named objects. GCL commands also accept geometry in these forms.

The general syntax is

```
<oogl-object>  ::=
       [ "{" ]
           [ "define" symbolname ]
           [ ["="] object-keyword ...
                | "<" filename
                | ":" symbolname ]
       [ "}" ]
```

where "quoted" items are literal strings (which appear without the quotes), [bracketed] items are optional, and `|` denotes alternatives. Curly braces, when present, must match; the outermost set of curly braces is generally required when the object is in a larger context, e.g. when it is part of a larger object or embedded in a Geomview command stream.

For example, each of the following three lines:

```
{ define fred   QUAD 1 0 0  0 0 1  0 1 0  1 0 0 }

{ define fred = QUAD 1 0 0  0 0 1  0 1 0  1 0 0 }

{ appearance { +edge } LIST { < "file1" } { : fred } }

VECT 1 2 0   2 0   0 0 0   1 1 2
```

is a valid OOGL object. The last example is only valid when it is delimited unambiguously by residing in its own disk file.

The `":"` construct allows references to symbols, created with `define`. A symbol's initial value is a null object. When a symbol is (re)defined, all references to it are automatically changed.

The `"define` NAME" construct allows to define a global symbol for the given object. If "NAME" already references an object, then the old object is discarded and replaced by the new definition. See Section 7.2.111 [read], page 126. See Section 7.2.59 [hdefine], page 116.

The `"<"` construct causes a disk file to be read. Note that this isn't a general textual "include" mechanism; a complete OOGL object must appear in the referenced file.

Files read using `"<"` are sought first in the directory of the file which referred to them, if any; failing that, the normal search path (see Section 7.2.72 [load-path], page 119) is used. The default search looks first in the current directory, then in the Geomview data directories.

Again, white space and line breaks are insignificant, and "#" comments may appear anywhere.

4.1.10 Appearances

Geometric objects can have associated "appearance" information, specifying shading, lighting, color, wire-frame vs. shaded-surface display, and so on. Appearances are inherited through object hierarchies, e.g. attaching an appearance to a `LIST` means that the appearance is applied to all the `LIST`'s members.

Some appearance-related properties are relegated to "material", "lighting" and "texture" substructures. Take care to note which properties belong to which structure. Any geometric object can be preceded by an appearance definition like in the following example:

```
{
   appearance { +edge }
   LIST { < "file1" } { QUAD 1 0 0  0 0 1  0 1 0  1 0 0 }
}
```

Appearances are also OOGL objects in their own right and can be given symbolic names and referenced by them (see Section 4.1.9 [References], page 52). See Section 4.3.1 [appearance objects], page 72.

Texture Mapping

There is a separate section concerning the definition of textures (see Section 4.1.11 [Texture Mapping], page 56).

Transparency

Rendering translucent objects is not supported by all drawing back ends. The OpenGL renderer has limited support for it: top-level objects (i.e. those which appear in the object browser of the main panel (see Section 3.3 [Basic Interaction], page 25) are rendered correctly by means of alpha-blending). Also, the RenderMan snapshots will include opacity values.

Here's an example appearance structure including values for all attributes. Order of attributes is unimportant. As usual, white space is irrelevant. Boolean attributes may be preceded by "+" or "-" to turn them on or off; "+" is assumed if only the attribute name appears. Other attributes expect values.

A "*" prefix on any attribute, e.g. "*+edge" or "*linewidth 2" or "material { *diffuse 1 1 .25 }", selects "override" status for that attribute.

```
appearance {
  +face               # (Do) draw faces of polygons.  On by default.
  -edge               # (Don't) draw edges of polygons
  +vect               # (Do) draw VECTs.  On by default.
  transparent screendoor
                      # (Enable) transparency. Enabling transparency
                      # does not (necessarily) result in a correct Geomview
                      # pictures, but alpha values are used in RenderMan
                      # snapshots.
                      # The allowed keywords are ``screendoor'' (masking out
                      # out pixels by means of a stipple pattern),
```

```
                    # ``blending'' for alpha-blending with BSP-tree
                    # space paritioning and depth-sorting (slow) and
                    # ``naive'' for alpha-blending without even
                    # depth-sorting, not to talk about space
                    # partioning. Omitting the keyword will default to
                    # alpha-blending with BSP-tree space-partioning
                    # and depth-sorting.
-normal             # (Do) draw surface-normal vectors
normscale 1         # ... with length 1.0 in object coordinates

+evert              # do evert polygon normals where needed so as
                    #   to always face the camera

+texturing          # (Enable) texture mapping
+linear             # (Enable) linear average of closest texture elements
+mipmap             # (Enable) texture mip-mapping
+mipinterp          # (Enable) linear mip-mapping
-backcull           # (Don't) discard clockwise-oriented faces
-concave            # (Don't) expect and handle concave polygons
-shadelines         # (Don't) shade lines as if they were lighted cylinders
                    # These four are only effective where the graphics system
                    # supports them, namely on GL and Open GL.

-keepcolor          # Normally, when N-D positional coloring is enabled as
                    # with geomview's (ND-color ...) command, all
                    # objects' colors are affected.  But, objects with the
                    # "+keepcolor" attribute are immune to N-D coloring.

shading smooth      # or ``shading constant'' or ``shading flat'' or
                    # or ``shading csmooth'' or ``shading vcflat''.
                    # smooth = Gouraud shading, flat = faceted,
                    # csmooth = smoothly interpolated but unlighted,
                    # vcflat = flat shading, but smoothly interpolated colors.

linewidth 1         # lines, points, and edges are 1 pixel wide.

patchdice 10 10     # subdivide Bezier patches this finely in u and v

material {         # Here's a material definition;
                    # it could also be read from a file as in
                    #  ``material < file.mat''

    ka  1.0         # ambient reflection coefficient.
    ambient .3 .5 .3 # ambient color (red, green, blue components)
                    # The ambient contribution to the shading is
                    # the product of ka, the ambient color,
                    # and the color of the ambient light.
```

```
    kd  0.8         # diffuse-reflection coefficient.
    diffuse .9 1 .4 # diffuse color.
                      # (In ``shading constant'' mode, the surface
                      # is colored with the diffuse color.)

    ks 1.0          # specular reflection coefficient.
    specular 1 1 1  # specular (highlight) color.
    shininess  25   # specular exponent; larger values give
                    # sharper highlights.

    backdiffuse .7 .5 0 # back-face color for two-sided surfaces
                      # If defined, this field determines the diffuse
                      # color for the back side of a surface.
                      # It's implemented by the software shader, and
                      # by hardware shading on GL systems which support
                      # two-sided lighting, and under Open GL.

    alpha   1.0     # opacity; 0 = transparent (invisible), 1 = opaque.
                    # Ignored when transparency is disabled.

    edgecolor   1 1 0  # line & edge color

    normalcolor 0 0 0  # color for surface-normal vectors
}

lighting {          # Lighting model

    ambient  .3 .3 .3  # ambient light

    replacelights   # ``Use only the following lights to
                    # illuminate the objects under this
                    # appearance.''
                    # Without "replacelights", any lights listed
                    # are added to those already in the scene.

                    # Now a collection of sample lights:
    light {
        color  1 .7 .6      # light color
        position  1 0 .5 0  # light position [distant light]
                            # given in homogeneous coordinates.
                            # With fourth component = 0,
                            # this means a light coming from
                            # direction (1,0,.5).
    }

    light {                         # Another light.
```

```
            color 1 1 1
            position  0 0 .5 1  # light at finite position ...
            location camera     # specified in camera coordinates.
                                # (Since the camera looks toward -Z,
                                # this example places the light
                                # .5 unit behind the eye.)
            # Possible "location" keywords:
            #  global    light position is in world (well, universe) coordinates
            #             This is the default if no location specified.
            #  camera   position is in the camera's coordinate system
            #  local    position is in the coordinate system where
            #                   the appearance was defined
        }
    }                   # end lighting model
    texture {
          clamp st               # or ``s'' or ``t'' or ``none''
          file lump.tiff         # file supplying texture-map image
          alphafile mask.pgm.Z   # file supplying transparency-mask image
          apply blend            # or ``modulate'' or ``decal''
          transform  1 0 0 0     # surface (s,t,0,1) * tfm -> texture coords
                     0 1 0 0
                     0 0 1 0
                    .5 0 0 1

          background 1 0 0 1     # relevant for ``apply blend''
    }
  }                     # end appearance
```

There are rules for inheritance of appearance attributes when several are imposed at different levels in the hierarchy.

For example, Geomview installs a backstop appearance which provides default values for most parameters; its control panels install other appearances which supply new values for a few attributes; user-supplied geometry may also contain appearances.

The general rule is that the child's appearance (the one closest to the geometric primitives) wins. Further, appearance controls with "override" status (e.g. `*+face` or `material { *diffuse 1 1 0 }`) win over those without it.

Geomview's appearance controls use the "override" feature so as to be effective even if user-supplied objects contain their own appearance settings. However, if a user-supplied object contains an appearance field with override status set, that property will be immune to Geomview's controls.

4.1.11 Texture Mapping

Some rendering back-ends support texture-mapped objects, actually only the OpenGL and the RenderMan interface at the time of this writing. There are also some issues with the RMan interface when using an alpha-channel in the texture image. Those rendering back-ends which don't support texturing silently ignore attempts to use texture mapping. A texture is specified as part of an appearance structure (see Section 4.1.10 [Appearances],

page 53). Briefly, one provides a texture image (see Section 4.3.2 [image], page 72), which is considered to lie in a square in `(s,t)` parameter space in the range 0 <= s <= 1, 0 <= t <= 1. Then one provides a geometric primitive, with each vertex tagged with `(s,t)` texture coordinates. If texturing is enabled, the appropriate portion of the texture image is pasted onto each face of the textured object.

There is (currently) no provision for inheritance of part of a texture structure; if the `texture` keyword is mentioned in an appearance, it supplants any other texture specification.

The appearance attribute `texturing` controls whether textures are used; there's no performance penalty for having texture { ... } fields defined when texturing is off.

The available fields are:

```
clamp   none  -or-  s  -or-  t  -or-  st
  Determines the meaning of texture coordinates outside the range 0..1.
  With clamp none, the default, coordinates are interpreted
  modulo 1, so (s,t) = (1.25,0), (.25,0), and (-.75,0) all refer to
  the same point in texture space.  With s or t or
  st, either or both of s- or t-coordinates less than 0 or
  greater than 1 are clamped to 1 or 0, respectively.

image { <image specification> (see Section 4.3.2 [image], page 72) }
  Specify the actual texture image. Images can have 1, 2, 3 or 4 channels:
    1 channel:  luminance
    2 channels: luminance and alpha
    3 channels: RGB data
    4 channels: RGBA data

  See Section 4.3.2 [image], page 72, for the actual definition of image
  objects. The alpha-channel is only interpreted as mask: where the mask
  is zero, pixels are simply not drawn. An exception is the case where
  apply is equal to modulate and translucency is enabled:
  in this case the resulting alpha value is the result of the
  multiplication of the surface color with the alpha value of the
  texture's alpha channel.

file      filename
alphafile filename
  This is considered obsolete, and only kept for compatibility,
  the modern way is to use the new OOGL image object. See Section 4.3.2
[image], page 72.
  The stuff documented here should still work, though

  Specifies image file(s) containing the texture.

  The file keyword specifies a file with color or lightness information;
  alphafile if present, specifies a transparency ("alpha") mask;
  where the mask is zero, pixels are simply not drawn.
```

```
  Several image file formats are available; the file type must be
  indicated by the last few characters of the file name:
    .ppm or .ppm.Z or .ppm.gz  24-bit 3-color image in PPM format
    .pgm or .pgm.Z or .pgm.gz  8-bit grayscale image in PGM format
    .sgi or .sgi.Z or .sgi.gz  8-bit, 24-bit, or 32-bit SGI image
    .tiff                      8-bit or 24-bit TIFF image
    .gif                       GIF image
  For this feature to work, some programs must be available in
  geomview's search path:
    zcat  for .Z files
    gzip  for .gz files
    tifftopnm for .tiff files
    giftoppm for .gif files

  If an alphafile image is supplied, it must be the same size
  as the file image.

  Image objects provide a more flexible way to specify texture
  data. See Section 4.3.2 [image], page 72.

apply   modulate  -or-  blend  -or-  decal
  Indicates how the texture image is applied to the surface.
  Here the "surface color" means the color the surface would have
  in the absence of texture mapping.

  With modulate, the default, the texture color (or lightness,
  if textured by a gray-scale image) is multiplied by the surface color.

  With blend, texture blends between the background color
  and the surface color.  The file parameter must specify a
  gray-scale image.  Where the texture image is 0, the surface color is
  unaffected; where it's 1, the surface is painted in the color given
  by background; and color is interpolated for intermediate values.

  With decal, the file parameter must specify a
  3-color image.  If an alphafile parameter is present,
  its value interpolates between the surface color (where alpha=0)
  and the texture color (where alpha=1).  Lighting does not affect the
  texture color in decal mode; effectively the texture is
  constant-shaded.

background  R G B A
  Specifies a 4-component color, with R, G, B, and A floating-point
  numbers normally in the range 0..1, used when apply blend
  is selected.

transform transformation-matrix
```

```
Expects a list of 16 numbers, or one of the other ways of representing
a transformation (: handlename or < filename).
The 4x4 transformation matrix is applied to texture coordinates,
in the sense of a 4-component row vector (s,t,0,1) multiplied on
the left of the matrix, to produce new coordinates (s',t')
which actually index the texture.
```

4.2 Object File Formats

4.2.1 QUAD: collection of quadrilaterals

The conventional suffix for a `QUAD` file is `.quad`.

The file syntax is

```
[C][N][4]QUAD  -or-  [C][N][4]POLY              # Key word
vertex  vertex  vertex  vertex  # 4*N vertices for some N
vertex  vertex  vertex  vertex
...
```

The leading key word is `[C][N][4]QUAD` or `[C][N][4]POLY`, where the optional `C` and `N` prefixes indicate that each vertex includes colors and normals respectively. That is, these files begin with one of the words

```
QUAD CQUAD NQUAD CNQUAD POLY CPOLY NPOLY CNPOLY
```

(but not `NCQUAD` or `NCPOLY`). `QUAD` and `POLY` are synonymous; both forms are allowed just for compatibility with ChapReyes.

Following the key word is an arbitrary number of groups of four vertices, each group describing a quadrilateral. See the Vertex syntax above. The object ends at end-of-file, or with a close-brace if incorporated into an object reference (see above).

A `QUAD BINARY` file format is accepted; See Section 4.1.8 [Binary format], page 51. The first word of binary data must be a 32-bit integer giving the number of quads in the object; following that is a series of 32-bit floats, arranged just as in the ASCII format.

4.2.2 MESH: rectangularly-connected mesh

The conventional suffix for a `MESH` file is `.mesh`.

The file syntax is

```
[U][C][N][Z][4][u][v][n]MESH # Key word
[Ndim]                  # Space dimension, present only if nMESH
Nu Nv             # Mesh grid dimensions
                            # Nu*Nv vertices, in format specified
                            # by initial key word
vertex(u=0,v=0)  vertex(1,0)  ... vertex(Nu-1,0)
vertex(0,1) ...    vertex(Nu-1,1)
...
vertex(0,Nv-1) ... vertex(Nu-1,Nv-1)
```

The key word is `[U][C][N][Z][4][u][v][n]MESH`. The optional prefix characters mean:

'U' Each vertex includes a 3-component texture space parameter. The first two components are the usual S and T texture parameters for that vertex; the third should be specified as zero.

'C' Each vertex (see Vertices above) includes a 4-component color.

'N' Each vertex includes a surface normal vector.

'Z' Of the 3 vertex position values, only the Z component is present; X and Y are omitted, and assumed to equal the mesh (u,v) coordinate so X ranges from 0 .. (Nu-1), Y from 0 .. (Nv-1) where Nu and Nv are the mesh dimensions – see below.

'4' Vertices are 4D, each consists of 4 floating values. Z and 4 cannot both be present.

'u' The mesh is wrapped in the u-direction, so the (0,v)'th vertex is connected to the (*Nu*-1,v)'th for all v.

'v' The mesh is wrapped in the v-direction, so the (u,0)'th vertex is connected to the (u,*Nv*-1)'th for all u. Thus a u-wrapped or v-wrapped mesh is topologically a cylinder, while a uv-wrapped mesh is a torus.

'n' Specifies a mesh whose vertices live in a higher dimensional space. The dimension follows the "MESH" keyword. Each vertex then has *Ndim* components.

Note that the order of prefix characters is significant; a colored, u-wrapped mesh is a CuMESH not a uCMESH.

Following the mesh header are integers *Nu* and *Nv*, the dimensions of the mesh.

Then follow *Nu***Nv* vertices, each in the form given by the header. They appear in v-major order, i.e. if we name each vertex by (u,v) then the vertices appear in the order

```
(0,0) (1,0) (2,0) (3,0) ...  (Nu-1,0)
(0,1) (1,1) (2,1) (3,1) ...  (Nu-1,1)
...
(0,Nv-1)                 ...  (Nu-1,Nv-1)
```

A MESH BINARY format is accepted; See Section 4.1.8 [Binary format], page 51. The values of *Nu* and *Nv* are 32-bit integers; all other values are 32-bit floats.

4.2.3 BBOX: simple bounding boxes

This is a very simple toy-object: it takes 2 vertices and draws a (hyper-) cube which is the bounding box of the two vertices.

Syntax:

```
BBOX
x[0] y[0] z[0]
x[1] y[1] z[1]
```

or

```
4BBOX
x[0] y[0] z[0] w[0]
x[1] y[1] z[1] w[1]
```

or

```
nBBOX
Ndim # > 3
x[0] y[0] z[0] w[0] ...
x[1] y[1] z[1] w[1] ...
```

or

```
4nBBOX
Ndim # > 3
d[0] x[0] y[0] z[0] w[0] ...
d[0] x[1] y[1] z[1] w[1] ...
```

There is no BBOX binary format. The 4 modifyer has different meanings depending on the dimension of the bounding box: `4BBOX` means that the 4 components of the vertices make up a 4-dimensional bounding-box. Using `4` in conjunction with `n` – `4nBBOX` *NDim* – means that the vertices specified in the file have *NDim+1* components, but the component at index 0 is the homogeneous divisor (in contrast to the ordinary 3d case where the homogeneous divisor would be the `w` – the third – component).

4.2.4 Bezier Surfaces

The conventional file suffixes for Bezier surface files are `.bbp` or `.bez`. A file with either suffix may contain either type of patch.

Syntax:

```
[ST]BBP -or- [C]BEZ<Nu><Nv><Nd>[_ST]
                        # Nu, Nv are u- and v-direction
                        # polynomial degrees in range 1..6
                        # Nd = dimension: 3->3-D, 4->4-D (rational)
                        # (The '<' and '>' do not appear in the input.)
                        # Nu,Nv,Nd are each a single decimal digit.
                        # BBP form implies Nu=Nv=Nd=3 so BBP = BEZ333.

                # Any number of patches follow the header
                        # (Nu+1)*(Nv+1) patch control points
                        # each 3 or 4 floats according to header
vertex(u=0,v=0)  vertex(1,0) ... vertex(Nu,0)
vertex(0,1)                          ... vertex(Nu,1)
...
vertex(0,Nv)                    ... vertex(Nu,Nv)

                        # ST texture coordinates if mentioned in header
S(u=0,v=0)     T(0,0)   S(0,Nv) T(0,Nv)
S(Nu,0)  T(Nu,0)    S(Nu,Nv) T(Nu,Nv)

                        # 4-component float (0..1) R G B A colors
                        # for each patch corner if mentioned in header
RGBA(0,0)   RGBA(0,Nv)
RGBA(Nu,0)  RGBA(Nu,Nv)
```

These formats represent collections of Bezier surface patches, of degrees up to 6, and with 3-D or 4-D (rational) vertices.

The header keyword has the forms `[ST]BBP` or `[C]BEZ<`*Nu*`><`*Nv*`><`*Nd*`>[_ST]` (the '<' and '>' are not part of the keyword.

The `ST` prefix on `BBP`, or `_ST` suffix on `BEZuvn`, indicates that each patch includes four pairs of floating-point texture-space coordinates, one for each corner of the patch.

The `C` prefix on `BEZuvn` indicates a colored patch, including four sets of four-component floating-point colors (red, green, blue, and alpha) in the range 0..1, one color for each corner.

Nu and *Nv*, each a single digit in the range 1..6, are the patch's polynomial degree in the u and v direction respectively.

Nd is the number of components in each patch vertex, and must be either `3` for 3-D or `4` for homogeneous coordinates, that is, rational patches.

`BBP` patches are bicubic patches with 3-D vertices, so `BBP` = `BEZ333` and `STBBP` = `BEZ333_ST`.

Any number of patches follow the header. Each patch comprises a series of patch vertices, followed by optional (s,t) texture coordinates, followed by optional (r,g,b,a) colors.

Each patch has (*Nu*+1)*(*Nv*+1) vertices in v-major order, so that if we designate a vertex by its control point indices (u,v) the order is

```
(0,0) (1,0) (2,0) ...  (Nu,0)
(0,1) (1,1) (2,1) ...  (Nu,1)
...
(0,Nv)             ...  (Nu,Nv)
```

with each vertex containing either 3 or 4 floating-point numbers as specified by the header.

If the header calls for ST coordinates, four pairs of floating-point numbers follow: the texture-space coordinates for the (0,0), (*Nu*,0), (0,*Nv*), and (*Nu*,*Nv*) corners of the patch, respectively.

If the header calls for colors, four four-component (red, green, blue, alpha) floating-point colors follow, one for each patch corner.

The series of patches ends at end-of-file, or with a closebrace if incorporated in an object reference.

4.2.5 OFF Files

The conventional suffix for `OFF` files is `.off`.

Syntax:

```
[ST][C][N][4][n]OFF     # Header keyword
[Ndim]            # Space dimension of vertices, present only if nOFF
NVertices  NFaces  NEdges   # NEdges not used or checked

x[0]  y[0]  z[0]      # Vertices, possibly with normals,
                        # colors, and/or texture coordinates, in that order,
                        # if the prefixes N, C, ST
                        # are present.
                        # If 4OFF, each vertex has 4 components,
```

```
                        # including a final homogeneous component.
                        # If nOFF, each vertex has Ndim components.
                        # If 4nOFF, each vertex has Ndim+1 components.
...
x[NVertices-1]  y[NVertices-1]  z[NVertices-1]

                        # Faces
                        # Nv = # vertices on this face
                        # v[0] ... v[Nv-1]: vertex indices
                        #               in range 0..NVertices-1
Nv  v[0] v[1] ... v[Nv-1]  colorspec
...
                        # colorspec continues past v[Nv-1]
                        # to end-of-line; may be 0 to 4 numbers
                        # nothing: default
                        # integer: colormap index
                        # 3 or 4 integers: RGB[A] values 0..255
                        # 3 or 4 floats: RGB[A] values 0..1
```

`OFF` files (name for "object file format") represent collections of planar polygons with possibly shared vertices, a convenient way to describe polyhedra. The polygons may be concave but there's no provision for polygons containing holes.

An `OFF` file may begin with the keyword `OFF`; it's recommended but optional, as many existing files lack this keyword.

Three ASCII integers follow: *NVertices*, *NFaces*, and *NEdges*. These are the number of vertices, faces, and edges, respectively. Current software does not use nor check *NEdges*; it needn't be correct but must be present.

The vertex coordinates follow: dimension * *Nvertices* floating-point values. They're implicitly numbered 0 through *NVertices*-1. dimension is either 3 (default) or 4 (specified by the key character `4` directly before `OFF` in the keyword).

Following these are the face descriptions, typically written with one line per face. Each has the form

```
N  Vert1 Vert2 ... VertN  [color]
```

Here *N* is the number of vertices on this face, and *Vert1* through *VertN* are indices into the list of vertices (in the range 0..*NVertices*-1).

The optional *color* may take several forms. Line breaks are significant here: the *color* description begins after *VertN* and ends with the end of the line (or the next # comment). A *color* may be:

nothing
: the default color

one integer
: index into "the" colormap; see below

three or four integers
: RGB and possibly alpha values in the range 0..255

three or four floating-point numbers
: RGB and possibly alpha values in the range 0..1

For the one-integer case, the colormap is currently read from the file `cmap.fmap` in Geomview's `data` directory. Some better mechanism for supplying a colormap is likely someday.

The meaning of "default color" varies. If no face of the object has a color, all inherit the environment's default material color. If some but not all faces have colors, the default is gray (R,G,B,A=.666).

A `[ST][C][N][n]OFF BINARY` format is accepted; See Section 4.1.8 [Binary format], page 51. It resembles the ASCII format in almost the way you'd expect, with 32-bit integers for all counters and vertex indices and 32-bit floats for vertex positions (and texture coordinates or vertex colors or normals if `COFF/NOFF/CNOFF/STCNOFF`/etc. format).

Exception: each face's vertex indices are followed by an integer indicating how many color components accompany it. Face color components must be floats, not integer values. Thus a colorless triangular face might be represented as

```
int int int int int
3   17   5   9   0
```

while the same face colored red might be

```
int int int int int float float float float
 3  17   5   9   4   1.0   0.0   0.0   1.0
```

4.2.6 VECT Files

The conventional suffix for `VECT` files is `.vect`.

Syntax:

```
[4]VECT
NPolylines  NVertices  NColors

Nv[0] ... Nv[NPolylines-1]     # number of vertices
                               # in each polyline

Nc[0] ... Nc[NPolylines-1]     # number of colors supplied
                               # in each polyline

Vert[0] ... Vert[NVertices-1]  # All the vertices
                               # (3*NVertices floats)

Color[0] ... Color[NColors-1]  # All the colors
                               # (4*NColors floats, RGBA)
```

`VECT` objects represent lists of polylines (strings of connected line segments, possibly closed). A degenerate polyline can be used to represent a point.

A `VECT` file begins with the key word `VECT` or `4VECT` and three integers: *NLines*, *NVertices*, and *NColors*. Here *NLines* is the number of polylines in the file, *NVertices* the total number of vertices, and *NColors* the number of colors as explained below.

Next come *NLines* **16-bit** integers

Nv[0] Nv[1] Nv[2] . . . Nv[NLines-1]

giving the number of vertices in each polyline. A negative number indicates a closed polyline; 1 denotes a single-pixel point. The sum (of absolute values) of the *Nv[i]* must equal *NVertices*.

Next come *NLines* more **16-bit** integers *Nc[i]*: the number of colors in each polyline. Normally one of three values:

0 No color is specified for this polyline. It's drawn in the same color as the previous polyline.

1 A single color is specified. The entire polyline is drawn in that color.

abs(*Nv[i]*) Each vertex has a color. Either each segment is drawn in the corresponding color, or the colors are smoothly interpolated along the line segments, depending on the implementation.

Next come *NVertices* groups of 3 or 4 floating-point numbers: the coordinates of all the vertices. If the keyword is *4VECT* then there are 4 values per vertex. The first abs(*Nv[0]*) of them form the first polyline, the next abs(*Nv[1]*) form the second and so on.

Finally *NColors* groups of 4 floating-point numbers give red, green, blue and alpha (opacity) values. The first *Nc[0]* of them apply to the first polyline, and so on.

A *VECT BINARY* format is accepted; see Section 4.1.8 [Binary format], page 51. The binary format exactly follows the ASCII format, with 32-bit Big-Endian integers where ordinary integer numbers appear, and with **16-bit** Big-Endian integers where 16-bit integers appear; 32-bit Big-Endian floats where real values appear. **BIG FAT NOTE:** The vertex counts *Nv[i]* and the color counts *Nc[i]* are **16-bit** Big-Endian integers.

4.2.7 SKEL Files

`SKEL` files represent collections of points and polylines, with shared vertices. The conventional suffix for `SKEL` files is `.skel`.

Syntax:

```
[C][4][n]SKEL
[NDim]                    # Vertex dimension, present only if nSKEL
NVertices  NPolylines

x[0]  y[0]  z[0]  # Vertices
                                  # if 4SKEL, each vertex has 4 components
                                  # if nSKEL, each vertex has NDim components
                                  # if C[4][n]SKEL vertex coordinates are
                                  # followed by an RGBA color specification
...
x[NVertices-1]  y[NVertices-1]  z[NVertices-1]

                        # Polylines
                        # Nv = vertices on this polyline (1 = point)
                        # v[0] ... v[Nv-1]: vertex indices
                        #               in range 0..NVertices-1
Nv  v[0] v[1] ... v[Nv-1]  [colorspec]
...
```

```
# colorspec continues past v[Nv-1]
# to end-of-line; may be nothing, or 3 or 4 numbers.
# nothing: default color
# 3 or 4 floats: RGB[A] values 0..1
```

The syntax resembles that of `OFF` files, with a table of vertices followed by a sequence of polyline descriptions, each referring to vertices by index in the table. Each polyline has an optional color.

For `nSKEL` objects, each vertex has *NDim* components. For `4nSKEL` objects, each vertex has *NDim+1* components; the final component is the homogeneous divisor.

A *[4][n]SKEL BINARY* format is accepted; See Section 4.1.8 [Binary format], page 51. It resembles the ASCII format in almost the way you'd expect, with 32-bit integers for all counters and vertex indices and 32-bit floats for vertex positions.

Exception: each polyline's vertex indices are followed by an integer indicating how many color components accompany it. Polyline color components must be floats, not integer values. Thus a colorless polyline with 3 vertices might be represented as

```
int int int int int
3   17   5   9   0
```

while the same polyline colored red might be

```
int int int int int float float float float
 3  17   5   9   4   1.0   0.0   0.0   1.0
```

4.2.8 SPHERE Files

The conventional suffix for `SPHERE` files is `.sph`.

```
[ST][E|H|S]SPHERE # Keyword
# auto-generated texture co-ordinates, only allowed with STSPHERE objects
[SINUSOIDAL|CYLINDRICAL|RECTANGULAR|STEREOGRAPHIC|ONEFACE]
# next four fields are required
Radius
Xcenter Ycenter Zcenter
```

The key word is `[ST][E|H|S]SPHERE`. The optional prefix characters mean:

'`ST`' The sphere carries automatically generated texture co-ordinates. See below.

'`E`' The sphere lives in Euclidean space.

'`H`' The sphere lives in Hyperbolic space. See Chapter 8 [Non-Euclidean Geometry], page 138.

'`S`' The sphere lives in spherical space. See Chapter 8 [Non-Euclidean Geometry], page 138.

Sphere objects are drawn using meshes which are rectangular in a polar co-ordinate system, with the equatorial plane parallel to the `x,y`-plane. Their smoothness, and the time taken to draw them, depends on the setting of the dicing level, 10x10 by default. From Geomview, the Appearance panel, the `<N>ad` keyboard command, or a `dice nu nv` Appearance attribute sets this.

Texture co-ordinates are generated for `STSPHERE` objects; the keyword following the initial `STSPHERE` keyword defines the way this is done. It follows the conventions of the `mktxmesh` Perl-script which comes with the *Orrery*.

SINUSOIDAL
: sinusoidal equal-area projection

CYLINDRICAL
: cylindrical proj: s is the longitude, t is the latitude

RECTANGULAR
: rectangular proj: s is the longitude, t is `sin(latitude)` (i.e. `z` co-ordinate in the sphere's co-ordinate system)

STEREOGRAPHIC
: stereographic projection from the south (`z=-1`) pole

ONEFACE
: stretch orthographic view of `+y` hemisphere over both, mirroring

4.2.9 INST Files

The conventional suffix for a `INST` file is `.inst`.

There is no INST BINARY format.

An `INST` applies a 4x4 (or *(N+1)*x*(N+1)* in the context of ND-viewing) transformation to another OOGL object. It begins with `INST` followed by these sections which may appear in any order:

`geom` *`oogl-object`*

specifies the OOGL object to be instantiated. See Section 4.1.9 [References], page 52, for the syntax of an *oogl-object*. The keyword `unit` is a synonym for `geom`.

```
transform   ["{"] 4x4 transform ["}"]
```

specifies a single transformation matrix. Either the matrix may appear literally as 16 numbers, or there may be a reference to a "transform" object, i.e.

```
"<" file-containing-4x4-matrix
```

or

```
":" symbol-representing-transform-object
```

Another way to specify the transformation is

```
transforms
    oogl-object
```

The *oogl-object* must be a `TLIST` object (list of transformations) object, or a `LIST` whose members are ultimately `TLIST` objects. In effect, the `transforms` keyword takes a collection of 4x4 matrices and replicates the `geom` object, making one copy for each 4x4 matrix.

If no `transform` nor `transforms` keyword appears, no transformation is applied (actually the identity is applied). You could use this for, e.g., wrapping an appearance around an externally-supplied object, though a single-membered LIST would do this more efficiently.

See Section 4.1.6 [Transformation matrices], page 51, for the matrix format.

When replicating a single geometry by means of a `TLIST` object (see '`transforms`' above) it may be useful to transform texture co-ordinates by another list of transformations; that list can be specified by

```
txtransforms
    TLIST-object
```

The number of texture transformations must match the number of geometry transformations. The `SPHERE` object (see Section 4.2.8 [SPHERE], page 66) uses this technique to generate an entire textured sphere out of some fraction of a sphere (usually one octant).

A single *(N+1)*-dimensional transformation can be specified by

```
ntransform ["{"] N+1 N+1 (N+1)x(N+1) floats ["}"]
```

This gives a single *N+1*-dimensional transformation matrix. Either the matrix may appear literally as *(N+1)*x*(N+1)* numbers, or there may be a reference to an '`ntransform`' object, i.e.

```
"<" file-containing-(N+1)x(N+1)-matrix
```

or

```
":" symbol-representing-ntransform-object
```

See Section 4.1.7 [ND Transformation matrices], page 51, for the matrix format.

Two more INST fields are accepted: `location` and `origin`.

Note that `location` as well as `origin` are ignored if this `INST` object carries an `ntransform`. Also, if ND-viewing is active (`ND-axes` command, see Chapter 7 [GCL], page 107) then `INST` objects with `origin` unequal to `local` will not be drawn, though the `location` stuff may work (or not).

```
location [global or camera or ndc or screen or local]
```

Normally an INST specifies a position relative to its parent object; the `location` field allows putting an object elsewhere.

- `location global` attaches the object to the global (a.k.a. universe) coordinate system – the same as that in which geomview's World objects, alien geometry, and cameras are placed.
- `location camera` places the object relative to the camera. (Thus if there are multiple views, it may appear in a different spatial position in each view.) The center of the camera's view is along its negative Z axis; positive X is rightward, positive Y upward. Normally the units of camera space are the same as global coordinates. When a camera is reset, the global origin is at (0,0,-3.0).
- `location ndc` places the object relative to the normalized unit cube into which the camera's projection (perspective or orthographic) maps the visible world. X, Y, and Z are each in the range from -1 to +1, with Z = -1 the near and Z = +1 the far clipping plane, and X and Y increasing rightward and upward respectively. Thus something like

  ```
  INST  transform  1 0 0 0  0 1 0 0  0 0 1 0  -.9 -.9 -.999 1
        location ndc
        geom < label.vect
  ```

  pastes `label.vect` onto the lower left corner of each window, and in front of nearly everything else, assuming `label.vect`'s contents lie in the positive quadrant of the X-Y plane. It's tempting to use -1 rather than -.999 as the Z component of the position, but

that may put the object just nearer than the near clipping plane and make it (partially) invisible, due to floating-point error.

- `location screen` places the object in screen coordinates. The range of Z is still -1 through +1 as for ndc coordinates; X and Y are measured in pixels, and range from (0,0) at the *lower left* corner of the window, increasing rightward and upward.

`location local` is the default; the object is positioned relative to its parent.

```
origin [global or camera or ndc or screen or local] x y z
```

The `origin` field translates the contents of the INST to place the origin at the specified point of the given coordinate system. Unlike `location`, it doesn't change the orientation, only the choice of origin. Both `location` and `origin` can be used together.

So for example

```
{ INST
  location screen
  origin ndc 0 0 -.99
  geom { < xyz.vect }
  transform { 100 0 0 0  0 100 0 0  0 0 -.009 0   0 0 0 1 }
}
```

places xyz.vect's origin in the center of the window, just beyond the near clipping plane. The unit-length X and Y edges are scaled to be just 100 screen units – pixels – long, regardless of the size of the window.

4.2.9.1 INST Examples

Here are some examples of `INST` files

```
INST
     unit < xyz.vect
     transform {
        1 0 0 0
        0 1 0 0
        0 0 1 0
        1 3 0 1
     }
{ appearance { +edge  material { edgecolor 1 1 0 } }
    INST geom < mysurface.quad }
{INST transform {: T} geom {<dodec.off}}
{ INST
     transforms
         { LIST
        { < some-matrices.prj }
        { < others.prj }
        { TLIST <still more of them> }

         }
     geom
         { # stuff replicated by all the above matrices
```

```
        ...
         }
}
```

This one resembles the `origin` example in the section above, but makes the X and Y edges be 1/4 the size of the window (1/4, not 1/2, since the range of ndc X and Y coordinates is -1 to +1).

```
{ INST
  location ndc
  geom { < xyz.vect }
  transform { .5 0 0 0  0 .5 0 0  0 0 -.009 0   0 0 -.99 1 }
}
```

4.2.10 LIST Files

The conventional suffix for a `LIST` file is `.list`.

A list of OOGL objects

Syntax:

```
LIST
    oogl-object
    oogl-object
    ...
```

Note that there's no explicit separation between the oogl-objects, so they should be enclosed in curly braces ({ }) for sanity. Likewise there's no explicit marker for the end of the list; unless appearing alone in a disk file, the whole construct should also be wrapped in braces, as in:

```
{ LIST { QUAD ... } { < xyz.quad } }
```

A `LIST` with no elements, i.e. `{ LIST }`, is valid, and is the easiest way to create an empty object. For example, to remove a symbol's definition you might write

```
{ define somesymbol  { LIST } }
```

4.2.11 TLIST Files

The conventional suffix for a `TLIST` file is `.grp` ("group") or `.prj` ("projective" matrices).

Collection of 4x4 matrices, used in the `transforms` section of and `INST` object.

Syntax:

```
TLIST                    # key word
<4x4 matrix (16 floats)>
...                   # any number of 4x4 matrices
transform {              # reference to a transform object
<transform object (can be a handle)>
}
tlist {                  # nested TLIST
<TLIST OOGL object (can be a handle)>
}
```

`TLIST`s are used only within the `transforms` clause of an `INST` object. They cause the `INST`s `geom` object to be instantiated once under each of the transforms in the `TLIST`. The

effect is like that of a `LIST` of `INST`s each with a single transform, and all referring to the same object, but is more efficient.

`TLIST`s can be nested: effectively this means that all transformations in each nested `TLIST` object are multiplied (from the left) by the transformations in the outer `TLIST` object.

Be aware that a `TLIST` is a kind of geometry object, distinct from a `transform` object. Some contexts expect one type of object, some the other. For example in

```
INST transform { : myT } geom { ... }
```

myT must be a transform object, which might have been created with the GCL

```
(read transform { define myT 1 0 0 1 ... })
```

while in

```
INST transforms { : myTs } geom { ... }
or
INST transforms { LIST {: myTs} {< more.prj} } geom { ... }
```

myTs must be a geometry object, defined e.g. with

```
(read geometry { define myTs { TLIST 1 0 0 1 ... } })
```

A `TLIST BINARY` format is accepted. Binary data begins with a 32-bit integer giving the number of transformations, followed by that number of 4x4 matrices in 32-bit floating-point format. The order of matrix elements is the same as in the ASCII format.

4.2.12 GROUP Files

This format is obsolete, but is still accepted. It combined the functions of `INST` and `TLIST`, taking a series of transformations and a single Geom (`unit`) object, and replicating the object under each transformation.

```
GROUP ... < matrices > ... unit { oogl-object }
```

is still accepted and effectively translated into

```
INST
        transforms { TLIST ... <matrices> ... }
        unit { oogl-object }
```

4.2.13 DISCGRP Files

This format is for discrete groups, such as appear in the theory of manifolds or in symmetry patterns. This format has its own man page. See discgrp(5).

4.2.14 COMMENT Objects

The COMMENT object is a mechanism for encoding arbitrary data within an OOGL object. It can be used to keep track of data or pass data back and forth between external modules.

Syntax:

```
COMMENT                 # key word

name type   # individual name and type specifier
{ ... }             # arbitrary data
```

The data, which must be enclosed by curly braces, can include anything except unbalanced curly braces. The *type* field can be used to identify data of interest to a particular program through naming conventions.

`COMMENT` objects are intended to be associated with other objects through inclusion in a `LIST` object. (See Section 4.2.10 [LIST], page 70.) The "#" OOGL comment syntax does not suffice for data exchange since these comments are stripped when an OOGL object is read in to Geomview. The `COMMENT` object is preserved when loaded into Geomview and is written out intact.

Here is an example associating a WorldWide Web URL with a piece of geometry:

```
{ LIST
 { < Tetrahedron}
 {COMMENT GCHomepage HREF { http://www.geomview.org/ }}
}
```

A binary `COMMENT` format is accepted. Its format is not consistent with the other OOGL binary formats. See Section 4.1.8 [Binary format], page 51. The `name` and `type` are followed by

```
N Byte1 Byte2 ... ByteN
```

instead of data enclosed in curly braces.

4.3 Non-geometric objects

The syntax of these objects is given in the form used in See Section 4.1.9 [References], page 52, where "quoted" items should appear literally but without quotes, square bracketed ([ ]) items are optional, and | separates alternative choices.

4.3.1 Appearance Objects

Appearances are OOGL objects of their own right, which simply means that it is possible to give them symbolic names (see Section 4.1.9 [References], page 52). There are other sections dealing with appearance details. See Section 4.1.10 [Appearances], page 53.

4.3.2 Image Objects

Image objects are used to specify pixmap data for either textures (see Section 4.1.11 [Texture Mapping], page 56), or for background images of cameras (see Section 7.2.19 [camera], page 111).

At the time of this writing images are comprised of 1 to 4 channels, a channel provides a single number in the range from 0 to `maxval` for each pixel; and `maxval` is tied to 255. The interpretation of the image data depending on the number of channels is like follows:

#Channels	Channel No.	Interpretation
1	1	greyscale or luminance data
2	1	greyscale or luminance data
	2	alpha channel (0: transparent, `maxval`: opaque)
3	1	red channel
	2	green channel
	3	blue channel
4	1	red channel

2	green channel
3	blue channel
4	alpha channel (0: transparent, `maxval`: opaque)

Image data can be specified inline (embedded into the current data stream) or via file references; in both cases the data is read in and interpreted at the time the image object is parsed. *This is different from the old (and deprecated) texture image specification, where the the image data on-disk would eventually be re-read by Geomview.*

The general syntax of image objects is like follows:

```
<image> ::=
  [ "{" ]              (curly brace, generally needed to make
                        the end of the object unambiguous.)
  [ "image" ]          (optional keyword; unnecessary if the type
                        is determined by the context, which it
                        usually is.)
  [ "define" <name> ]
                       (defines an image named <name>, setting
                       its value from the stuff which follows)
  |
      "<" <filename>   (meaning: read image from that file)
  |
      ":" <name>       (meaning: use variable name,
                       defined elsewhere; if undefined the image
                       carries no data)
  |
                       (actual stuff defining the image; image data
                       must come last, after defining the width and
                       height and number of channels)

      "width"          (width of the image, auto-detected from image data
                       if possible)

      "height"         (height of the image, auto-detected from image data
                       if possible)

      "channels"       (number of channels, auto-detected from image data
                       and data specifications, if possible)

      "maxval"         (unsupported, must be 255 if specified)

      "data DESTMASK [FILTER] [{] < FILENAME [}]"
      "data DESTMASK [FILTER] DATASIZE [{][\n]LITERAL_IMAGE_DATA[}]"
                       (either external or embedded image data, see below
                       for a detailed description of the meaning of
                       MASK and FILTER. An image can (and
                       has, in general) multiple data sections.)
```

```
[ "}" ]              (matching curly brace)
```

Details concerning the image data specification:

'DESTMASK'

This is a bit-field describing where the specified image data should be placed in the destination pixmap. The bit-field is specified by an integer in one of the known formats (decimal, octal, hexadecimal). The channels of the source data are always enumbered consecutively. If, e.g. '`FILENAME`' or '`LITERAL_IMAGE_DATA`' specify a three-channel (probably RGB . . .) image and '`DESTMASK`' equals `0xD` (i.e. bit 1 is 0), then the 3rd channel of the source pixmap would be placed in the 4th channel of the destination image object (the alpha channel), the 2nd source channel would determine the '`blue`' destination value and the 1st source channel the '`red`' destination value.

The number of channels of the source data always has to match the number of bits set in '`DESTMASK`'. **Exception:** if the source pixmap has only one channel, then it can be used to fill any number of destination channels; all channels specified in '`DESTMASK`' are filled with the data of the single channel source pixmap.

Geomview knows the following symbolic constants, which can be used instead of specifying the '`DESTMASK`' bit-field numerically:

`LUMINANCE`
same as '`1`', '`0x1`', '`\01`'

`LUMINANCE_ALPHA`
same as '`3`', '`0x3`', '`\03`'

`RGB` same as '`7`', '`0x7`', '`\07`'

`RGBA` same as '`15`', '`0xf`', '`\017`'

`ALPHA` depends on the context: the absolute number of channels must be known; i.e. '`data ALPHA ...`' must be prepended by something determining the number of channels of the image, e.g.

```
...
data RGB ...
data ALPHA
...
```

is valid, but

```
<no other channel or image data specification>
data ALPHA ...
<whatever else ...>
```

is not valid, because Geomview has not means to determine the destination channel from the context.

`AUTO` PGM image data is interpreted as single grey-scale channel, RGB PNM data as RGB image data. `AUTO` cannot work with '`raw`' image data.

'FILTER' The 'FILTER' specification is optional. If it is missing, then Geomview tries to determine the image type from the 'FILENAME' suffix. If there is no suffix or the suffix is unknown, or for embedded image data, Geomview is able to auto-detectc SGI image file formats (for historical reasons . . .) and NetPBM image formats (for practical reasons). The auto-detection of NetPBM formats includes the new *PAM* image format which allows (among other things) to store an alpha channel together with the luminance or RGB data. Likewise, the final output of any of the specified filters must either be in SGI image file format, or specify a PAM, PNM or PGM image. If the image file format cannot be determined by either the filenmae suffix or the filter specification or by auto-deteciong of SGI or NetPBM data, then Geomview assume that it is raw-data. See below.

The decompression filters may be prepended to either one of the know image formats or an explicitly specified filter, e.g. the following is valid:

```
data LUMINANCE raw.gzip { < gzippedgreymapfile }
```

The above should be equivalent to

```
data LUMINANCE raw { < greymapfile },
```

provided that the decompressed data carries single channel data, with the first pixel corresponding to the lower-left corner (because of the 'raw' image format, see below).

Geomview has builtin knowledge for the following filters/suffixes:

Data Decompression

'z'

'gz'

'gzip' data is piped through 'gzip -dc'

'bz2'

'bzip2' data is piped through 'bzip2 -dc'

Image Formats

'tiff'

'tif' TIFF image file format. Only supported if the tifftopnm executable can be fond in the current excution path.

'png' PNG image file format. Only supported if the pngtopnm executable can be fond in the current excution path.

'jpg'

'jpeg' JPEG image file format. Only supported if the jpegtopnm executable can be fond in the current excution path.

'gif' GIF image file format. Only supported if the giftoppm executable can be fond in the current excution path.

'raw' Raw image data; the number of channels must match the number of bits set in 'DESTMASK'. Pixels are specified with 1 byte per channel. The pixels are organized in rows as 'liminance[-alpha]' or 'RGB[A]' samples. The left-most pixel is the first pixel in each data-row, the top-most image row must come first (this is just the same as the NetPBM convention, the image coordinate systems has its origin in the left/top corner, as usual).

Explicitly Specified Filters

If none of the suffixes specified above should match, then the suffix/filter is interpreted as an external filter program; the filter program must read from STDIN and write to STDOUT. The output must either be in SGI image format, or in PNM or PGM format. Otherwise the output data is interpreted as raw image data (see above).

Something like the following should work, provided that the program ${HOME}/bin/bububfilter exists, is executable and does something useful:

```
...
data RGB "${HOME}/bin/bububfilter.bzip2" 7 { # binary data follows
bububub
}
...
```

Note that – prior to feeding the data to the 'bububfilter' – Geomview will try to decompress the stuff with 'bzip2 -dc'.

Missing image data: Normally, the number of image channels is determined automatically from the image data specifications; if the image specification carries an explicit number of channels via the channels keyword which exceeds the number of channels found in the image data specifications, or if the union of all 'DESTMASK' specfications has holes, then missing luminance and RGB channels are initialized to 0, and a missing alpha-channel is initialized to maxval, i.e. omitting the alpha channel data for an RGBA image is just the same as defining an RGB image.

4.3.3 Transform Objects

Where a single 4x4 matrix is expected – as in the INST transform field, the camera's camtoworld transform and the Geomview xform* commands – use a transform object.

Note that a transform is distinct from a TLIST, which is a type of geometry. TLISTs can contain one or more 4x4 transformations; "transform" objects must have exactly one.

Why have both? In many places – e.g. camera positioning – it's only meaningful to have a single transform. Using a separate object type enforces this.

Syntax for a transform object is

```
<transform> ::=
  [ "{" ]              (curly brace, generally needed to make
                        the end of the object unambiguous.)
```

```
   [ "transform" ]    (optional keyword; unnecessary if the type
                       is determined by the context, which it
                       usually is.)
   [ "define" <name> ]
                      (defines a transform named <name>, setting
                       its value from the stuff which follows)

      <sixteen floating-point numbers>
                      (interpreted as a 4x4 homogeneous transform
                       given row by row, intended to apply to a
                       row vector multiplied on its LEFT, so that e.g.
                       Euclidean translations appear in the bottom row)
   |
      "<" <filename>  (meaning: read transform from that file)
   |
      ":" <name>      (meaning: use variable <name>,
                        defined elsewhere; if undefined the initial
                        value is the identity transform)

 [ "}" ]              (matching curly brace)
```

The whole should be enclosed in { braces }. Braces are not essential if exactly one of the above items is present, so e.g. a 4x4 array of floats standing alone may but needn't have braces.

Some examples, in contexts where they might be used:

```
# Example 1: A GCL command to define a transform
# called "fred"

(read transform { transform  define fred
         1 0 0 0
         0 1 0 0
         0 0 1 0
        -3 0 1 1
    }
)

# Example 2:  A camera object using transform
# "fred" for camera positioning
# Given the definition above, this puts the camera at
# (-3, 0, 1), looking toward -Z.

{ camera
        halfyfield 1
        aspect 1.33
        camtoworld { : fred }
}
```

4.3.4 ND-Transform Objects

Where – in the context of ND-viewing – a single (N+1)x(N+1) matrix is expected – as in the `INST ntransform` field, or the `ND-xform*` (see Chapter 7 [GCL], page 107) commands – use an `ntransform` object.

`ntransform` are *NRows* x *NCols* transformation matrix where usually *NRows* = *N+1* in the context of *N*-dimensional objects and viewing. The homogeneous component of an `ntransform` sits at column zero (in contrast to ordinary `transform` objects where it is located at column three). `ntransform` objects operate on points of any dimension: if a point is to be transformed by an `ntransform` object and the dimension of the point does not match the number of rows of the `ntransform` object, then either the point is implicitly padded with zeros to match *NRows* or the matrix is implicitly padded with ones down its diagonal (and zeros everywhere else) such that it will operate as identity on the excess dimensions of the input point.

Syntax for an `ntransform` object is

```
<ntransform> ::=
  [ "{" ]              (curly brace, generally needed to make
                        the end of the object unambiguous.)

   [ "ntransform" ]    (optional keyword; unnecessary if the type
                        is determined by the context, which it
                        usually is.)

   [ "define" <name> ]
                        (defines a transform named <name>, setting
                        its value from the stuff which follows)

     NRows NCols
                        (number of rows and columns of the matrix,
                        typically N+1 N+1, but any dimensions
                        are possible)
     <NRows x NCols floating-point numbers>
                        (interpreted as a NRows x NCols
                        homogeneous transform given row by row, intended
                        to apply to a row vector multiplied on its LEFT,
                        so that e.g. Euclidean translations appear in the
                        bottom row -- in contrast to the ordinary
                        transform objects where the translations appear
                        in the bottom row)
   |
      "<" <filename>   (meaning: read transform from that file)
   |
      ":" <name>       (meaning: use variable <name>,
                       defined elsewhere; if undefined the initial
                       value is the identity transform)

 [ "}" ]              (matching curly brace)
```

The whole should be enclosed in { braces }. Braces are not necessarily essential, so e.g. two integers – *NRows NCols* – followed by a *NRows* x *NCols* array of floats standing alone may but needn't have braces.

Some examples, in contexts where they might be used:

```
# Example 1: A GCL command to define a 6x6 transform called
# "fred", a mere translation by the vector -3 0 1 1 0. This
# transform is meant for a five dimensional space, with an homogeneous
# component a index zero.

(read ntransform { ntransform  define fred
         6 6
         1 -3 0 1 1 0
         0  1 0 0 0 0
         0  0 1 0 0 0
         0  0 0 1 0 0
         0  0 0 0 1 0
         0  0 0 0 0 1
    }
)
# Example 2: Set the ND-xform of an object -- a geometry or a camera
# cluster. Given the definition above, this puts the object at (-3 0 1 1
# 0) in the five dimensional space.

(ND-xform-set focus : fred)

# or

(ND-xform-set g1 : fred)
```

4.3.5 cameras

A camera object specifies the following properties of a camera:

position and orientation
: specified by either a camera-to-world or world-to-camera transformation; this transformation does not include the projection, so it's typically just a combination of translation and rotation. Specified as a transform object, typically a 4x4 matrix.

"focus" distance
: Intended to suggest a typical distance from the camera to the object of interest; used for default camera positioning (the camera is placed at (X,Y,Z) = (0,0,focus) when reset) and for adjusting field-of-view when switching between perspective and orthographic views.

window aspect ratio
: True aspect ratio in the sense <Xsize>/<Ysize>. This normally should agree with the aspect ratio of the camera's window. Geomview normally adjusts the aspect ratio of its cameras to match their associated windows.

near and far clipping plane distances
: Note that both must be strictly greater than zero. Very large <far>/<near> distance ratios cause Z-buffering to behave badly; part of an object may be visible even if somewhat more distant than another.

field of view
: Specified in either of two forms.

  'fov' is the field of view – in degrees if perspective, or linear distance if orthographic – in the *shorter* direction.

  'halfyfield'
  : is half the projected Y-axis field, in world coordinates (not angle!), at unit distance from the camera. For a perspective camera, halfyfield is related to angular field:

    halfyfield = tan(Y_axis_angular_field / 2)

    while for an orthographic one it's simply:

    halfyfield = Y_axis_linear_field / 2

  This odd-seeming definition is (a) easy to calculate with and (b) well-defined in both orthographic and perspective views.

background color
: Arguably not a property of a camera, but of the scene. Nevertheless, as there is no "background" OOGL object, and the background color should not be a property of the drawing device, it can be specified here. At the time of this writing, however, the GUI always overrides the background color with its own settings.

background image
: Same reasoning as above, only that the GUI does not override this setting. The image is centered at NDC co-ordinates (0,0,-1); it is not resized, just painted behind everything else as is. See Section 4.3.2 [image], page 72.

The syntax for a camera is:

```
<camera> ::=

   [ "camera" ]                 (optional keyword)
    [ "{" ]                    (opening brace, generally required)
        [ "define" <name> ]

        "<" <filename>
      |
        ":" <name>
      |
                                (or any number of the following,
                                 in any order...)

        "perspective"  {"0" | "1"}            (default 1)
                                          (otherwise orthographic)
```

```
        "stereo"       {"0" | "1"}            (default 0)
                                          (otherwise mono)

        "worldtocam" <transform>        (see transform syntax above)

        "camtoworld" <transform>
                                 (no point in specifying both
                                  camtoworld and worldtocam; one is
                                  constrained to be the inverse of

        "halfyfield" <half-linear-Y-field-at-unit-distance>
                                 (default tan 40/2 degrees)

        "fov"           (angular field-of-view if perspective,
                         linear field-of-view otherwise.
                         Measured in whichever direction is smaller,
                         given the aspect ratio.  When aspect ratio
                         changes -- e.g. when a window is reshaped --
                         "fov" is preserved.)

        "frameaspect" <aspect-ratio>    (X/Y) (default 1.333)

        "near"  <near-clipping-distance>        (default 0.1)

        "far"   <far-clipping-distance>         (default 10.0)

        "focus" <focus-distance>                (default 3.0)

        "bgcolor" <float RGB(A) color>          (default 1/3 1/3 1/3 1)

        "bgimage" { <image specification> }     (default no background image)

      [ "}" ]                            (matching closebrace)
```

4.3.6 window

A window object specifies size, position, and other window-system related information about a window in a device-independent way.

The syntax for a window object is:

```
window ::=

        [ "window" ]                     (optional keyword)
          [ "{" ]                       (curly brace, often required)

                                 (any of the following, in any order)
```

```
"size"  <xsize> <ysize>
                (size of the window)

"position"  <xmin> <xmax> <ymin> <ymax>
                (position & size)


"noborder"
                (specifies the window should
                 have no window border)

"pixelaspect"  <aspect>
            (specifies the true visual aspect ratio
             of a pixel in this window in the sense
             xsize/ysize, normally 1.0.
             For stereo hardware which stretches the
             display vertically by a factor of 2,
             ``pixelaspect 0.5'' might do.
             The value is used when computing the
             projection of a camera associated with
             this window.)

[ "}" ]                     (matching closebrace)
```

Window objects are used in the Geomview `window` and `ui-panel` commands to set default properties for future windows or to change those of an existing window. See Section 7.2.156 [window], page 135. See Section 7.2.149 [ui-panel], page 134.

5 Customization: `.geomview` files

When Geomview is started, it loads and executes commands in a system-wide startup file named `.geomview`. This file is in the `data` subdirectory of the Geomview distribution directory and contains GCL commands to configure Geomview in a way common to all users on the system.

Next, Geomview looks for the file `~/.geomview` (`~` stands for your home directory). You can use this to configure your own default Geomview behavior to suit your tastes.

After reading `~/.geomview`, Geomview looks for a file named `.geomview` in the current directory. If such a file exists Geomview reads it, unless it is the same as `~/.geomview` (which would be the case if you are running Geomview from your home directory). You can use the current directory's `.geomview` to create a Geomview customization specific to a certain project.

You can use `.geomview` files to control all kinds of things about Geomview. They can contain any valid GCL statements. Especially useful is the `ui-panel` command which controls the initial placement of Geomview's panels. For an example see the system-wide `.geomview` file mentioned above. See Chapter 7 [GCL], page 107. See Section 7.2.149 [ui-panel], page 134.

It is a good idea to enclose all the commands you put in a `.geomview` file in a `progn` statement in order to cause Geomview to execute them all at once. Otherwise Geomview might execute them sequentially over the first few refresh cycles after starting up.

To change, e.g. the focus policy of the camera window such that they pick up the focus policy of the window manager (instead of being activated when the mouse cursor crosses the window), you could put the following in your `~/.geomview` file:

```
(progn
  (ui-cam-focus focus-change)
  ... # other stuff
)
```

You can put any valid `GCL` command into your `.geomview` files,see Chapter 7 [GCL], page 107. See Section 7.2.106 [progn], page 126. See Section 7.2.141 [ui-cam-focus], page 132.

6 External Modules

An external module is a program that interacts with Geomview. A module communicates with Geomview through GCL and can control any apsect of Geomview that you can control through Geomview's user interface.

In many cases an external module is a specialized program that implements some mathematical algorithm that creates a geometric object that changes shape as the algorithm progresses. The module informs Geomview of the new object shape at each step, so the object appears to evolve with time in the Geomview window. In this way Geomview serves as a *display engine* for the module.

An external module may be interactive. It can respond to mouse and keyboard events that take place in a Geomview window, thus extending the capability of Geomview itself.

6.1 How External Modules Interface with Geomview

External modules appear in the *Modules* browser in Geomview's *Main* panel. To run a module, click the left mouse button on the module's entry in the browser. While the module is running, an additional line for that module will appear in the browser. This line begins with a number in brackets, which indicates the *instace* number of the module. (For some modules it makes sense to have more than one instance of the module running at the same time.) You can kill an external module by clicking on its instance entry.

By default when Geomview starts, it displays all the modules that have been installed on your system.

For instructions on installing a module on your system so that it will appear in the *Modules* browser every time Geomview is run by anyone on your system, See Section 6.7 [Module Installation], page 105.

When Geomview invokes an external module, it creates pipes connected to the module's standard input and output. (Pipes are like files except they are used for communication between programs rather than for storing things on a disk.) Geomview interprets anything that the module writes to its standard output as a GCL command. Likewise, if the external module requests any data from Geomview, Geomview writes that data to the module's standard input. Thus all a module has to do in order to communicate with Geomview is write commands to standard output and (optionally) receive data on standard input. Note that this means that the module cannot use standard input and output for communicating with the user. If a module needs to communicate with the user it can do so either through a control panel of its own or else by responding to certain events that it finds out about from Geomview.

6.2 Example 1: Simple External Module

This section gives a very simple external module which displays an oscillating mesh. To try out this example, make a copy of the file `example1.c` (it is distributed with Geomview in the `doc` subdirectory) in your directory and compile it with the command

```
cc -o example1 example1.c -lm
```

Then put the line

```
(emodule-define "Example 1" "./example1")
```

in a file called `.geomview` in your current directory. Then invoke Geomview; it is important that you compile the example program, create the `.geomview` file, and invoke Geomview all in the same directory. You should see "Example 1" in the *Modules* browser of Geomview's *Main* panel; click on this entry in the browser to start the module. A surface should appear in your camera window and should begin oscillating. You can stop the module by clicking on the "[1] Example 1" line in the *Modules* browser.

```
/*
 * example1.c: oscillating mesh
 *
 * This example module is distributed with the Geomview manual.
 * If you are not reading this in the manual, see the "External
 * Modules" chapter of the manual for more details.
 *
 * This module creates an oscillating mesh.
 */

#include <math.h>
#include <stdio.h>

/* F is the function that we plot
 */
float F(x,y,t)
     float x,y,t;
{
  float r = sqrt(x*x+y*y);
  return(sin(r + t)*sqrt(r));
}

main(argc, argv)
     char **argv;
{
  int xdim, ydim;
  float xmin, xmax, ymin, ymax, dx, dy, t, dt;

  xmin = ymin = -5;             /* Set x and y            */
  xmax = ymax = 5;              /*    plot ranges         */
  xdim = ydim = 24;             /* Set x and y resolution */
  dt = 0.1;                     /* Time increment is 0.1  */

  /* Geomview setup.  We begin by sending the command
   *            (geometry example { : foo})
   * to Geomview.  This tells Geomview to create a geom called
   * "example" which is an instance of the handle "foo".
   */
  printf("(geometry example { : foo })\n");
  fflush(stdout);
```

```
  /* Loop until killed.
   */
  for (t=0; ; t+=dt) {
    UpdateMesh(xmin, xmax, ymin, ymax, xdim, ydim, t);
  }
}

/* UpdateMesh sends one mesh iteration to Geomview.  This consists of
 * a command of the form
 *    (read geometry { define foo
 *       MESH
 *       ...
 *    })
 * where ... is the actual data of the mesh.  This command tells
 * Geomview to make the value of the handle "foo" be the specified
 * mesh.
 */
UpdateMesh(xmin, xmax, ymin, ymax, xdim, ydim, t)
     float xmin, xmax, ymin, ymax, t;
     int xdim, ydim;
{
  int i,j;
  float x,y, dx,dy;

  dx = (xmax-xmin)/(xdim-1);
  dy = (ymax-ymin)/(ydim-1);

  printf("(read geometry { define foo \n");
  printf("MESH\n");
  printf("%1d %1d\n", xdim, ydim);
  for (j=0, y = ymin; j<ydim; ++j, y += dy) {
    for (i=0, x = xmin; i<xdim; ++i, x += dx) {
      printf("%f %f %f\t", x, y, F(x,y,t));
    }
    printf("\n");
  }
  printf("})\n");
  fflush(stdout);
}
```

The module begins by defining a function `F(x,y,t)` that specifies a time-varying surface. The purpose of the module is to animate this surface over time.

The main program begins by defining some variables that specify the parameters with which the function is to be plotted.

The next bit of code in the main program prints the following line to standard output

```
(geometry example { : foo })
```

This tells Geomview to create a geom called `example` which is an instance of the handle `foo`. *Handles* are a part of the OOGL file format which allow you to name a piece of geometry whose value can be specified elsewhere (and in this case updated many times); for more information on handles, See Chapter 4 [OOGL File Formats], page 49. In this case, `example` is the title by which the user will see the object in Geomview's object browser, and `foo` is the internal name of the handle that the object is a reference to.

We then do `fflush(stdout)` to ensure that Geomview receives this command immediately. In general, since pipes may be buffered, an external module should do this whenever it wants to be sure Geomview has actually received everything it has printed out.

The last thing in the main program is an infinite loop that cycles through calls to the procedure `UpdateMesh` with increasing values of `t`. `UpdateMesh` sends Geomview a command of the form

```
(read geometry { define foo
MESH
24 24
...
})
```

where `...` is a long list of numbers. This command tells Geomview to make the value of the handle `foo` be the specified mesh. As soon as Geomview receives this command, the geom being displayed changes to reflect the new geometry.

The mesh is given in the format of an OOGL MESH. This begins with the keyword `MESH`. Next come two numbers that give the x and y dimensions of the mesh; in this case they are both 24. This line is followed by 24 lines, each containing 24 triples of numbers. Each of these triples is a point on the surface. Then finally there is a line with `"})"` on it that ends the `"{"` which began the `define` statement and the `"("` that began the command. For more details on the format of MESH data, see Section 4.2.2 [MESH], page 59.

This module could be written without the use of handles by having it write out commands of the form

```
(geometry example {
MESH
24 24
...
})
```

This first time Geomview receives a command of this form it would create a geom called `example` with the given `MESH` data. Subsequent `(geometry example ...)` commands would cause Geomview to replace the geometry of the geom `example` with the new `MESH` data. If done in this way there would be no need to send the initial `(geometry example { : foo })` command as above. The handle technique is useful, however, because it can be used in more general situations where a handle represents only part of a complex geom, allowing an external module to replace only that part without having to retransmit the entire geom. For more information on handles, see Chapter 7 [GCL], page 107. See Section 4.1.9 [References], page 52. See Section 7.2.59 [hdefine], page 116. See Section 7.2.111 [read], page 126.

The module loops through calls to `UpdateMesh` which print out commands of the above form one after the other as fast as possible. The loop continues indefinitely; the module

will terminate when the user kills it by clicking on its instance line in the *Modules* browser, or else when Geomview exits.

Sometimes when you terminate this module by clicking on its instance entry the *Modules* browser, Geomview will kill it while it is in the middle of sending a command to Geomview. Geomview will then receive only a piece of a command and will print out a cryptic but harmless error message about this. When a module has a user interface panel it can use a "Quit" button to provide a more graceful way for the user to terminate the module. See the next example.

You can run this module in a shell window without Geomview to see the commands it prints out. You will have to kill it with *ctrl-C* to get it to stop.

6.3 Example 2: Simple External Module with FORMS Control Panel

This section gives a new version of the above module — one that includes a user interface panel for controlling the velocity of the oscillation. We use the FORMS library by Mark Overmars for the control panel. The FORMS library is a public domain user interface toolkit for IRISes.

To try out this example, make a copy of the file `example2.c` (distributed with Geomview in the `doc` subdirectory) in your directory and compile it with the command

```
cc -I/u/gcg/ngrap/include -o example2 example2.c \
  -L/u/gcg/ngrap/lib/sgi -lforms -lfm_s -lgl_s -lm
```

You should replace the string `/u/gcg/ngrap` above with the pathname of the Geomview distribution directory on your system. (The forms library is distributed with Geomview and the `-I` and `-L` options above tell the compiler where to find it.)

Then put the line

```
(emodule-define "Example 2" "./example2")
```

in a file called `.geomview` in the current directory and invoke Geomview from that directory. Click on the "Example 2" entry in the *Modules* browser to invoke the module. A small control panel should appear. You can then control the velocity of the mesh oscillation by moving the slider.

```
/*
 * example2.c: oscillating mesh with FORMS control panel
 *
 * This example module is distributed with the Geomview manual.
 * If you are not reading this in the manual, see the "External
 * Modules" chapter of the manual for an explanation.
 *
 * This module creates an oscillating mesh and has a FORMS control
 * panel that lets you change the speed of the oscillation with a
 * slider.
 */

#include <math.h>
#include <stdio.h>
```

```
#include <sys/time.h>           /* for struct timeval below */

#include "forms.h"              /* for FORMS library */

FL_FORM *OurForm;
FL_OBJECT *VelocitySlider;
float dt;

/* F is the function that we plot
 */
float F(x,y,t)
     float x,y,t;
{
  float r = sqrt(x*x+y*y);
  return(sin(r + t)*sqrt(r));
}

/* SetVelocity is the slider callback procedure; FORMS calls this
 * when the user moves the slider bar.
 */
void SetVelocity(FL_OBJECT *obj, long val)
{
  dt = fl_get_slider_value(VelocitySlider);
}

/* Quit is the "Quit" button callback procedure; FORMS calls this
 * when the user clicks the "Quit" button.
 */
void Quit(FL_OBJECT *obj, long val)
{
  exit(0);
}

/* create_form_OurForm() creates the FORMS panel by calling a bunch of
 * procedures in the FORMS library.  This code was generated
 * automatically by the FORMS designer program; normally this code
 * would be in a separate file which you would not edit by hand.  For
 * simplicity of this example, however, we include this code here.
 */
create_form_OurForm()
{
  FL_OBJECT *obj;
  FL_FORM *form;
  OurForm = form = fl_bgn_form(FL_NO_BOX,380.0,120.0);
  obj = fl_add_box(FL_UP_BOX,0.0,0.0,380.0,120.0,"");
  VelocitySlider = obj = fl_add_valslider(FL_HOR_SLIDER,20.0,30.0,
                                          340.0,40.0,"Velocity");
```

```
    fl_set_object_lsize(obj,FL_LARGE_FONT);
    fl_set_object_align(obj,FL_ALIGN_TOP);
    fl_set_call_back(obj,SetVelocity,0);
  obj = fl_add_button(FL_NORMAL_BUTTON,290.0,75.0,70.0,35.0,"Quit");
    fl_set_object_lsize(obj,FL_LARGE_FONT);
    fl_set_call_back(obj,Quit,0);
  fl_end_form();
}

main(argc, argv)
     char **argv;
{
  int xdim, ydim;
  float xmin, xmax, ymin, ymax, dx, dy, t;
  int fdmask;
  static struct timeval timeout = {0, 200000};

  xmin = ymin = -5;             /* Set x and y            */
  xmax = ymax = 5;              /*    plot ranges         */
  xdim = ydim = 24;             /* Set x and y resolution */
  dt = 0.1;                     /* Time increment is 0.1  */

  /* Forms panel setup.
   */
  foreground();
  create_form_OurForm();
  fl_set_slider_bounds(VelocitySlider, 0.0, 1.0);
  fl_set_slider_value(VelocitySlider, dt);
  fl_show_form(OurForm, FL_PLACE_SIZE, TRUE, "Example 2");


  /* Geomview setup.
   */
  printf("(geometry example { : foo })\n");
  fflush(stdout);

  /* Loop until killed.
   */
  for (t=0; ; t+=dt) {
    fdmask = (1 << fileno(stdin)) | (1 << qgetfd());
    select(qgetfd()+1, &fdmask, NULL, NULL, &timeout);
    fl_check_forms();
    UpdateMesh(xmin, xmax, ymin, ymax, xdim, ydim, t);
  }
}

/* UpdateMesh sends one mesh iteration to Geomview
```

```
 */
UpdateMesh(xmin, xmax, ymin, ymax, xdim, ydim, t)
     float xmin, xmax, ymin, ymax, t;
     int xdim, ydim;
{
  int i,j;
  float x,y, dx,dy;

  dx = (xmax-xmin)/(xdim-1);
  dy = (ymax-ymin)/(ydim-1);

  printf("(read geometry { define foo \n");
  printf("MESH\n");
  printf("%1d %1d\n", xdim, ydim);
  for (j=0, y = ymin; j<ydim; ++j, y += dy) {
    for (i=0, x = xmin; i<xdim; ++i, x += dx) {
      printf("%f %f %f\t", x, y, F(x,y,t));
    }
    printf("\n");
  }
  printf("})\n");
  fflush(stdout);
}
```

The code begins by including some header files needed for the event loop and the FORMS library. It then declares global variables for holding a pointer to the slider FORMS object and the velocity `dt`. These are global because they are needed in the slider callback procedure `SetVelocity`, which forms calls every time the user moves the slider bar. `SetVelocity` sets `dt` to be the new value of the slider.

`Quit` is the callback procedure for the *Quit* button; it provides a graceful way for the user to terminate the program.

The procedure `create_panel` calls a bunch of FORMS library procedures to set up the control panel with slider and button. For more information on using FORMS to create interface panels see the FORMS documentation. In particular, FORMS comes with a graphical panel designer that lets you design your panels interactively and generates code like that in `create_panel`.

This example's main program is similar to the previous example, but includes extra code to deal with setting up and managing the FORMS panel.

To set up the panel we call the GL procedure `foreground` to cause the process to run in the foreground. By default GL programs run in the background, and for various reasons external modules that use FORMS (which is based on GL) need to run in the foreground. We then call `create_panel` to create the panel and `fl_set_slider_value` to set the initial value of the slider. The call to `fl_show_form` causes the panel to appear on the screen.

The first three lines of the main loop, starting with

```
fdmask = (1 << fileno(stdin)) | (1 << qgetfd());
```

check for and deal with events in the panel. The call to `select` imposes a delay on each pass through the main loop. This call returns either after a delay of 1/5 second or when the next GL event occurs, or when data appears on standard input, whichever comes first. The `timeout` variable specifies the amount of time to wait on this call; the first member (0 in this example) gives the number of seconds, and the second member (200000 in this example) gives the number of microseconds. Finally, `fl_check_forms()` checks for and processes any FORMS events that have happened; in this case this means calling `SetVelocity` if the user has moved the slider or calling `Quit` if the user has clicked on the *Quit* button.

The purpose of the delay in the loop is to keep the program from using excessive amounts of CPU time running around its main loop when there are no events to be processed. This is not so crucial in this example, and in fact may actually slow down the animation somewhat, but in general with external modules that have event loops it is important to do something like this because otherwise the module will needlessly take CPU cycles away from other running programs (such as Geomview!) even when it isn't doing anything.

The last line of the main loop in this example, the call to `UpdateMesh`, is the same as in the previous example.

6.4 The XForms Library

XForms is a handy and relatively simple user interface toolkit for X11. Many Geomview external modules, including the examples in this manual, use XForms to create and manage control panels. XForms is available from `http://www.nongnu.org/xforms/`. XForms is free-ware. If you wish you may use any other interface toolkit instead of XForms in an external module. We chose FORMS because it is free and relatively simple.

There is a complete autoconf'ed package '`gvemod-xforms-example`' available from Geomview's Sourceforge.NET (`http://geomview.sourceforge.net`) page.

6.5 Example 3: External Module with Bi-Directional Communication

The previous two example modules simply send commands to Geomview and do not receive anything from Geomview. This section describes a module that communicates in both directions. There are two types of communication that can go from Geomview to an external module. This example shows *asynchronous* communication — the module needs to be able to respond at any moment to expressions that Geomview may emit which inform the module of some change of state within Geomview.

(The other type of communication is *synchronous*, where a module sends a request to Geomview for some piece of information and waits for a response to come back before doing anything else. The main GCL command for requesting information of this type is Section 7.2.158 [write], page 135. This module does not do any synchronous communication.)

In ansynchronous communication, Geomview sends expressions that are essentially echoes of GCL commands. The external module sends Geomview a command expressing interest in a certain command, and then every time Geomview executes that command, the module receives a copy of it. This happens regardless of who sent the command to Geomview; it can be the result of the user doing something with a Geomview panel, or it may have come from another module or from a file that Geomview reads. This is how a module can find out about and act on things that happen in Geomview.

This example uses the OOGL lisp library to parse and act on the expressions that Geomview writes to the module's standard input. This library is actually part of Geomview itself — we wrote the library in the process of implementing GCL. It is also convenient to use it in external modules that must understand a of subset of GCL — specifically, those commands that the module has expressed interest in.

This example shows how a module can receive user pick events, i.e. when the user clicks the right mouse button with the cursor over a geom in a Geomview camera window. When this happens Geomview generates an internal call to a procedure called `pick`; the arguments to the procedure give information about the pick, such as what object was picked, the coordinates of the picked point, etc. If an external module has expressed interest in calls to `pick`, then whenever `pick` is called Geomview will echo the call to the module's standard input. The module can then do whatever it wants with the pick information.

This module is the same as the *Nose* module that comes with Geomview. Its purpose is to illustrate picking. Whenever you pick on a geom by clicking the right mouse button on it, the module draws a little box at the spot where you clicked. Usually the box is yellow. If you pick a vertex, the box is colored magenta. If you pick a point on an edge of an object, the module will also highlight the edge by drawing cyan boxes at its endpoints and drawing a yellow line along the edge.

Note that in order for this module to actually do anything you must have a geom loaded into Geomview and you must click the right mouse button with the cursor over a part of the geom.

```
/*
 * example3.c: external module with bi-directional communication
 *
 * This example module is distributed with the Geomview manual.
 * If you are not reading this in the manual, see the "External
 * Modules" chapter of the manual for an explanation.
 *
 * This module is the same as the "Nose" program that is distributed
 * with Geomview.  It illustrates how a module can find out about
 * and respond to user pick events in Geomview.  It draws a little box
 * at the point where a pick occurrs.  The box is yellow if it is not
 * at a vertex, and magenta if it is on a vertex.  If it is on an edge,
 * the program also marks the edge.
 *
 * To compile:
 *
 *   cc -I/u/gcg/ngrap/include -g -o example3 example3.c \
 *      -L/u/gcg/ngrap/lib/sgi -loogl -lm
 *
 * You should replace "/u/gcg/ngrap" above with the pathname of the
 * Geomview distribution directory on your system.
 */

#include <stdio.h>
#include "lisp.h"               /* We use the OOGL lisp library */
```

```
#include "pickfunc.h"           /* for PICKFUNC below */
#include "3d.h"                 /* for 3d geometry library */

/* boxstring gives the OOGL data to define the little box that
 * we draw at the pick point.  NOTE:  It is very important to
 * have a newline at the end of the OFF object in this string.
 */
char boxstring[] = "\
INST\n\
transform\n\
.04 0 0 0\n\
0 .04 0 0\n\
0 0 .04 0\n\
0 0 0 1\n\
geom\n\
OFF\n\
8 6 12\n\
\n\
-.5 -.5 -.5     # 0   \n\
.5 -.5 -.5      # 1   \n\
.5  .5 -.5      # 2   \n\
-.5  .5 -.5     # 3   \n\
-.5 -.5  .5     # 4   \n\
.5 -.5  .5      # 5   \n\
.5  .5  .5      # 6   \n\
-.5  .5  .5     # 7   \n\
\n\
4 0 1 2 3\n\
4 4 5 6 7\n\
4 2 3 7 6\n\
4 0 1 5 4\n\
4 0 4 7 3\n\
4 1 2 6 5\n";

progn()
{
  printf("(progn\n");
}

endprogn()
{
  printf(")\n");
  fflush(stdout);
}

Initialize()
{
```

```
  extern LObject *Lpick();  /* This is defined by PICKFUNC below but must */
                            /* be used in the following LDefun() call */
  LInit();
  LDefun("pick", Lpick, NULL);

  progn(); {
    /* Define handle "littlebox" for use later
     */
    printf("(read geometry { define littlebox { %s }})\n", boxstring);

    /* Express interest in pick events; see Geomview manual for explanation.
     */
    printf("(interest (pick world * * * * nil nil nil nil nil))\n");

    /* Define "pick" object, initially the empty list (= null object).
     * We replace this later upon receiving a pick event.
     */
    printf("(geometry \"pick\" { LIST } )\n");

    /* Make the "pick" object be non-pickable.
     */
    printf("(pickable \"pick\" no)\n");

    /* Turn off normalization, so that our pick object will appear in the
     * right place.
     */
    printf("(normalization \"pick\" none)\n");

    /* Don't draw the pick object's bounding box.
     */
    printf("(bbox-draw \"pick\" off)\n");

  } endprogn();
}

/* The following is a macro call that defines a procedure called
 * Lpick().  The reason for doing this in a macro is that that macro
 * encapsulates a lot of necessary stuff that would be the same for
 * this procedure in any program.  If you write a Geomview module that
 * wants to know about user pick events you can just copy this macro
 * call and change the body to suit your needs; the body is the last
 * argument to the macro and is delimited by curly braces.
 *
 * The first argument to the macro is the name of the procedure to
 * be defined, "Lpick".
 *
 * The next two arguments are numbers which specify the sizes that
```

```
 * certain arrays inside the body of the procedure should have.
 * These arrays are used for storing the face and path information
 * of the picked object.  In this module we don't care about this
 * information so we declare them to have length 1, the minimum
 * allowed.
 *
 * The last argument is a block of code to be executed when the module
 * receives a pick event.  In this body you can refer to certain local
 * variables that hold information about the pick.  For details see
 * Example 3 in the Extenal Modules chapter of the Geomview manual.
 */
PICKFUNC(Lpick, 1, 1,
{
  handle_pick(pn>0, &point, vn>0, &vertex, en>0, edge);
},
/* version for picking Nd-objects (not documented here) */)

handle_pick(picked, p, vert, v, edge, e)
     int picked;               /* was something actually picked?     */
     int vert;                 /* was the pick near a vertex?        */
     int edge;                 /* was the pick near an edge?         */
     HPoint3 *p;               /* coords of pick point               */
     HPoint3 *v;               /* coords of picked vertex            */
     HPoint3 e[2];             /* coords of endpoints of picked edge */
{
  Normalize(&e[0]);            /* Normalize makes 4th coord 1.0 */
  Normalize(&e[1]);
  Normalize(p);
  progn(); {
    if (!picked) {
      printf("(geometry \"pick\" { LIST } )\n");
    } else {
      /*
       * Put the box in place, and color it magenta if it's on a vertex,
       * yellow if not.
       */
      printf("(xform-set pick { 1 0 0 0  0 1 0 0  0 0 1 0  %g %g %g 1 })\n",
             p->x, p->y, p->z);
      printf("(geometry \"pick\"\n");
      if (vert) printf("{ appearance { material { diffuse 1 0 1 } }\n");
      else printf("{ appearance { material { diffuse 1 1 0 } }\n");
      printf("  { LIST { :littlebox }\n");

      /*
       * If it's on an edge and not a vertex, mark the edge
       * with cyan boxes at the endpoins and a black line
       * along the edge.
```

```
        */
       if (edge && !vert) {
         e[0].x -= p->x; e[0].y -= p->y; e[0].z -= p->z;
         e[1].x -= p->x; e[1].y -= p->y; e[1].z -= p->z;
         printf("{ appearance { material { diffuse 0 1 1 } }\n\
  LIST\n\
   { INST transform 1 0 0 0 0 1 0 0 0 0 1 0 %f %f %f 1 geom :littlebox }\n\
   { INST transform 1 0 0 0 0 1 0 0 0 0 1 0 %f %f %f 1 geom :littlebox }\n\
   { VECT\n\
          1 2 1\n\
          2\n\
          1\n\
          %f %f %f\n\
          %f %f %f\n\
          1 1 0 1\n\
   }\n\
  }\n",
               e[0].x, e[0].y, e[0].z,
               e[1].x, e[1].y, e[1].z,
               e[0].x, e[0].y, e[0].z,
               e[1].x, e[1].y, e[1].z);
       }
       printf("    }\n  }\n)\n");
     }

  } endprogn();

}

Normalize(HPoint3 *p)
{
  if (p->w != 0) {
    p->x /= p->w;
    p->y /= p->w;
    p->z /= p->w;
    p->w = 1;
  }
}

main()
{
  Lake *lake;
  LObject *lit, *val;
  extern char *getenv();

  Initialize();
```

```
    lake = LakeDefine(stdin, stdout, NULL);
    while (!feof(stdin)) {

      /* Parse next lisp expression from stdin.
       */
      lit = LSexpr(lake);

      /* Evaluate that expression; this is where Lpick() gets called.
       */
      val = LEval(lit);

      /* Free the two expressions from above.
       */
      LFree(lit);
      LFree(val);
    }
  }
```

The code begins by defining procedures `progn()` and `endprogn()` which begin and end a Geomview `progn` group. The purpose of the Geomview `progn` command is to group commands together and cause Geomview to execute them all at once, without refreshing any graphics windows until the end. It is a good idea to group blocks of commands that a module sends to Geomview like this so that the user sees their cumulative effect all at once.

Procedure `Initialize()` does various things needed at program startup time. It initializes the lisp library by calling `LInit()`. Any program that uses the lisp library should call this once before calling any other lisp library functions. It then calls `LDefun` to tell the library about our `pick` procedure, which is defined further down with a call to the `PICKFUNC` macro. Then it sends a bunch of setup commands to Geomview, grouped in a `progn` block. This includes defining a handle called `littlebox` that stores the geometry of the little box. Next it sends the command

```
(interest (pick world * * * * nil nil nil nil nil))
```

which tells Geomview to notify us when a pick event happens.

The syntax of this `interest` statement merits some explanation. In general `interest` takes one argument which is a (parenthesized) expression representing a Geomview function call. It specifies a type of call that the module is interested in knowing about. The arguments can be any particular argument values, or the special symbols `*` or `nil`. For example, the first argument in the `pick` expression above is `world`. This means that the module is interested in calls to `pick` where the first argument, which specifies the coordinate system, is `world`. A `*` is like a wild-card; it means that the module is interested in calls where the corresponding argument has any value. The word `nil` is like `*`, except that the argument's value is not reported to the module. This is useful for cutting down on the amount of data that must be transmitted in cases where there are arguments that the module doesn't care about.

The second, third, fourth, and fifth arguments to the `pick` command give the name, pick point coordinates, vertex coordinates, and edge coordinates of a pick event. We specify these by `*`'s above. The remaining five arguments to the `pick` command give other information

about the pick event that we do not care about in this module, so we specify these with `nil`'s. For the details of the arguments to `pick`, See Chapter 7 [GCL], page 107.

The `geometry` statement defines a geom called `pick` that is initially an empty list, specified as `{ LIST }` ; this is the best way of specifying a null geom. The module will replace this with something useful by sending Geomview another `geometry` command when the user picks something. Next we arrange for the `pick` object to be non-pickable, and turn normalization off for it so that Geomview will display it in the size and location where we put it, rather than resizing and relocating it to fit into the unit cube.

The next function in the file, `Lpick`, is defined with a strange looking call to a macro called `PICKFUNC`, defined in the header file `pickfunc.h`. This is the function for handling pick events. The reason we provide a macro for this is that that macro encapsulates a lot of necessary stuff that would be the same for the pick-handling function in any program. If you write a Geomview module that wants to know about user pick events you can just copy this macro call and change it to suit yours needs.

In general the syntax for `PICKFUNC` is

```
PICKFUNC(name, block, NDblock)
```

where *name* is the name of the procedure to be defined, in this case `Lpick`. The next argument, *block*, is a block of code to be executed when a pick event occurs. If *block* contains a return statement, then the returned value must be a pointer to a Lisp-object, that is of type `LObject *`. The last argument has the same functionality as the *block* argument, but is only invoked when picking objects in a higher dimensional world.

`PICKFUNC` declares certain local variables in the body of the procedure. When the module receives a `(pick ...)` statement from Geomview, the procedure assigns values to these variables based on the information in the `pick` call (variables corresponding to `nil`'s in the `(interest (pick ...))` are not given values).

There is also a second variant of the `PICKFUNC` macro with a slightly different syntax:

```
DEFPICKFUNC(helpstr, coordsys, id,
  point, pn, vertex, vn, edge, en, face, fn, ppath, ppn,
  vi, ei, ein, fi,
  body, NDbody)
```

`DEFPICKFUNC` can be used as well as `PICKFUNC`, there is no functional difference with the exception that the name of the C-function is tied to `Lpick` when using `DEFPICKFUNC` and that the `(help pick)` GCL-command (see Section 7.2.61 [help], page 117) would respond with echoing *helpstr*.

The table below lists all variables defined in `PICKFUNC` In the context of ND-viewing `float` variants of the arguments apply: the *body* execution block sees the `HPoint3` variables, and the *NDbody* block sees only flat one-dimensional arrays of `float`-type.

In the ND-viewing context the co-ordinates passed to the pick function are still the 3-dimensional co-ordinates of the camera view-port where the pick occurred, but padded with zeroes on transformed back to the co-ordinate system specified by the second argument of the `pick` command.

`char *coordsys;`

> A string specifying the coordinate system in which coordinates are given. In this example, this will always be `world` because of the `interest` call above.

`char *id;` A string specifying the name of the picked geom.

`HPoint3 point; int pn;`
`float *point; int pn;`
: `point` is an `HPoint3` structure giving the coordinates of the picked point. `HPoint3` is a homogeneous point coordinate representation equivalent to an array of 4 floats. `pn` tells how many coordinates have been written into this array; it will always be either `0`, `4` or greater than `4`. If it is greater than `4`, then the *NDbody* instruction block is invoked and in this case `point` is a flat array of `pn` many `float`s. A value of zero means no point was picked, i.e. the user clicked the right mouse button while the cursor was not pointing at a geom. In this case the ordinary *block* 3d instruction block is executed.

`HPoint3 vertex; int vn;`
`float *vertex; int vn;`
: `vertex` is an `HPoint3` structure giving the coordinates of the picked vertex, if the pick point was near a vertex. `vn` tells how many coordinates have been written into this array; it will always be either `0` or greater equal `4`. A value of zero means the pick point was not near a vertex. In the context of ND-viewing `vertex` will be an array of `vn` `float`s and `vn` will be equal to `pn`.

`HPoint3 edge[2]; int en;`
`float *edge; int en;`
: `edge` is an array of two `HPoint3` structures giving the coordinates of the end-points of the picked edge, if the pick point was near an edge. `en` tells how many coordinates have been written into this array; it will always be `0` or greater equal `8`. A value of zero means the pick point was not near an edge. In the context of ND-viewing `edge` will be a flat one-dimensional array of `en` many `float`s: the first `pn` `float`s define the first vertex, and the second `pn` many `float`s define the second vertex; `en` will be two times `pn`.

In this example module, the remaining variables will never be given values because their values in the `interest` statement were specified as `nil`.

`HPoint3 face[]; int fn;`
`float *face; int fn;`
: `face` is a variable length array of *fn* `HPoint3`'s. `face` gives the coordinates of the vertices of the picked face. `fn` tells how many coordinates have been written into this array; it will always be either `0` or a multiple of `pn`. A value of zero means the pick point was not near a face. In the context of ND-viewing `face` is a flat one-dimensional array of `fn` many floats of which each vertex occupies `pn` many components.

`int ppath[]; int ppn;`
: `ppath` is an array of *maxpathlen* `int`'s. `ppath` gives the path through the OOGL heirarchy to the picked primitive. `pn` tells how many integers have been written into this array; it will be at most *maxpathlen*. A path of {3,1,2}, for example, means that the picked primitive is "subobject number 2 of subobject number 1 of object 3 in the world".

`int vi;` `vi` gives the index of the picked vertex in the picked primitive, if the pick point was near a vertex.

`int ei[2]; int ein`
The `ei` array gives the indices of the endpoints of the picked edge, if the pick point was near a vertex. `ein` tells how many integers were written into this array. It will always be either 0 or 2; a value of 0 means the pick point was not near an edge.

`int fi;` `fi` gives the index of the picked face in the picked primitive, if the pick point was near a face.

The `handle_pick` procedure actually does the work of dealing with the pick event. It begins by normalizing the homogeneous coordinates passed in as arguments so that we can assume the fourth coordinate is 1. It then sends GCL commands to define the `pick` object to be whatever is appropriate for the kind of pick recieved. See see Chapter 4 [OOGL File Formats], page 49, and see Chapter 7 [GCL], page 107, for an explanation of the format of the data in these commands.

The main program, at the bottom of the file, first calls `Initialize()`. Next, the call to `LakeDefine` defines the `Lake` that the lisp library will use. A `Lake` is a structure that the lisp library uses internally as a type of communiation vehicle. (It is like a unix stream but more general, hence the name.) This call to `LakeDefine` defines a `Lake` structure for doing I/O with `stdin` and `stdout`. The third argument to `LakeDefine` should be `NULL` for external modules (it is used by Geomview). Finally, the program enters its main loop which parses and evaluates expressions from standard input.

6.6 Example 4: Simple Tcl/Tk Module Demonstrating Picking

It's not necessary to write a Geomview module in C. The only requirement of an external module is that it send GCL commands to its standard output and expect responses (if any) on its standard input. An external module can be written in C, perl, tcl/tk, or pretty much anything.

As an example, assuming you have Tcl/Tk version 4.0 or later, here's an external module with a simple GUI which demonstrates interaction with geomview. This manual doesn't discuss the Tcl/Tk language; see the good book on the subject by its originator John Ousterhout, published by Addison-Wesley, titled *Tcl and the Tk Toolkit*.

The '`#!`' on the script's first line causes the system to interpret the script using the Tcl/Tk '`wish`' program; you might have to change its first line if that's in some location other than /usr/local/bin/wish4.0. Or, you could define it as a module using

```
(emodule-define  "Pick Demo"  "wish pickdemo.tcl")
```

in which case '`wish`' could be anywhere on the UNIX search path.

```
#! /usr/local/bin/wish4.0

# We use "fileevent" below to have "readsomething" be called whenever
# data is available from standard input, i.e. when geomview has sent us
# something.  It promises to include a trailing newline, so we can use
```

```
# "gets" to read the geomview response, then parse its nested parentheses
# into tcl-friendly {} braces.

proc readsomething {} {
  if {[gets stdin line] < 0} {
        puts stderr "EOF on input, exiting..."
        exit
  }
  regsub -all {\(} $line "\{" line
  regsub -all {\)} $line "\}" line
  # Strip outermost set of braces
  set stuff [lindex $line 0]
  # Invoke handler for whichever command we got.  Could add others here,
  # if we asked geomview for other kinds of data as well.
  switch [lindex $stuff 0] {
        pick     {handlepick $stuff}
        rawevent {handlekey $stuff}
  }
}

# Fields of a "pick" response, from geomview manual:
#     (pick COORDSYS GEOMID G V E F P VI EI FI)
#          The pick command is executed internally in response to pick
#          events (right mouse double click).
#
#          COORDSYS = coordinate system in which coordinates of the following
#              arguments are specified.   This can be:
#               world: world coord sys
#               self:  coord sys of the picked geom (GEOMID)
#               primitive: coord sys of the actual primitive within
#                   the picked geom where the pick occurred.
#          GEOMID = id of picked geom
#          G = picked point (actual intersection of pick ray with object)
#          V = picked vertex, if any
#          E = picked edge, if any
#          F = picked face
#          P = path to picked primitive [0 or more]
#          VI = index of picked vertex in primitive
#          EI = list of indices of endpoints of picked edge, if any
#          FI = index of picked face

# Report when user picked something.
#
proc handlepick {pick} {
  global nameof selvert seledge order
  set obj [lindex $pick 2]
  set xyzw [lindex $pick 3]
```

```
  set fv [lindex $pick 6]
  set vi [lindex $pick 8]
  set ei [lindex $pick 9]
  set fi [lindex $pick 10]

  # Report result, converting 4-component homogeneous point into 3-space point.
  set w [lindex $xyzw 3]
  set x [expr [lindex $xyzw 0]/$w]
  set y [expr [lindex $xyzw 1]/$w]
  set z [expr [lindex $xyzw 2]/$w]
  set s "$x $y $z "
  if {$vi >= 0} {
        set s "$s  vertex #$vi"
  }
  if {$ei != {}} {
        set s "$s  edge [lindex $ei 0]-[lindex $ei 1]"
  }
  if {$fi != -1} {
        set s "$s  face #$fi ([expr [llength $fv]/3]-gon)"
  }
  msg $s
}


# Having asked for notification of these raw events, we report when
# the user pressed these keys in the geomview graphics windows.

proc handlekey {event} {
  global lastincr
  switch [lindex $event 1] {
    32 {msg "Pressed space bar"}
     8 {msg "Pressed backspace key"}
  }
}


#
# Display a message on the control panel, and on the terminal where geomview
# was started.  We use ``puts stderr ...'' rather than simply ``puts ...'',
# since Geomview interprets anything we send to standard output
# as a GCL command!
#
proc msg {str} {
  global msgtext
  puts stderr $str
  set msgtext $str
  update
```

```
}

# Load object from file
proc loadobject {fname} {
  if {$fname != ""} {
        puts "(geometry thing < $fname)"
        # Be sure to flush output to ensure geomview receives this now!
        flush stdout
  }
}


# Build simple "user interface"

# The message area could be a simple label rather than an entry box,
# but we want to be able to use X selection to copy text from it.
# The default mouse bindings do that automatically.

entry .msg -textvariable msgtext -width 45
pack .msg

frame .f

  label .f.l -text "File to load:"
  pack .f.l -side left

  entry .f.ent -textvariable fname
  pack .f.ent -side left -expand true -fill x
  bind .f.ent <Return> { loadobject $fname }

pack .f


# End UI definition.


# Call "readsomething" when data arrives from geomview.

fileevent stdin readable {readsomething}

# Geomview initialization

puts {
        (interest (pick primitive))
        (interest (rawevent 32))        # Be notified when user presses space
        (interest (rawevent 8))         # or backspace keys.
        (geometry thing < hdodec.off)
```

```
        (normalization world none)
}
# Flush to ensure geomview receives this.
flush stdout

wm title . {Sample external module}

msg "Click right mouse in graphics window"
```

6.7 Module Installation

This section tells how to install an external module so you can invoke it within Geomview. There are two ways to install a module: you can install a *private* module so that the module is available to you whenever you run Geomview, or you can install a *system* module so that the module is available to all users on your system whenever they run Geomview.

6.7.1 Private Module Installation

The `emodule-define` command arranges for a module to appear in Geomview's *Modules* browser. The command takes two string arguments; the first is the name that will appear in the *Modules* browser. The second is the shell command for running the module; it may include arguments (see Section 7.2.40 [emodule-define], page 114). Geomview executes this command in a subshell when you click on the module's entry in the browser. For example

```
(emodule-define "Foo" "/u/home/modules/foo -x")
```

adds a line labeled `"Foo"` to the *Modules* browser which causes the command `"/u/home/modules/foo -x"` to be executed when selected.

You may put `emodule-define` commands in your `~/.geomview` file to arrange for certain modules to be available every time you run Geomview; See Chapter 5 [Customization], page 83. You can also execute `emodule-define` commands from the *Commands* panel to add a module to an already running copy of Geomview.

There are several other GCL commands for controlling the entries in the *Modules* browser; for details, See Chapter 7 [GCL], page 107.

6.7.2 System Module Installation

To install a module so that it is available to all Geomview users do the following

1. Create a file called `.geomview-`*`module`* where *`module`* is the name of the module. This file should contain a single line which is an `emodule-define` command for that module:

   ```
   (emodule-define "New Module" "newmodule")
   ```

   The first argument, `"New Module"` above, is the string that will appear in the *Modules* browser. The second string, `"newmodule"` above, is the Bourne shell command for invoking the module. It may include arguments, and you may assume that the module is on the $PATH searched by the shell.

2. Put a copy of the `.geomview-`*`module`* and the module executable itself in `/usr/local/libexec/geomview` directory.

After these steps, the new module should appear, in alphabetical position, in the *Modules* browser of Geomview's *Main* panel next time Geomview is run. The reason this works is that when Geomview is invoked it processes all the `.geomview-*` files in its `modules` directory. It also remembers the pathname of this directory and prepends that path to the $PATH of the shell in which it invokes such a module.

7 GCL: the Geomview Command Language

GCL has the syntax of lisp – i.e. an expression of the form (f a b . . .) means pass the values of a, b, . . . to the function f. GCL is very limited and is by no means an implementation of lisp. It is simply a language for expressing commands to be executed in the order given, rather than a programming language. It does not support variable or function definition.

GCL is the language that Geomview understands for files that it loads as well as for communication with other programs. To execute a GCL command interactively, you can bring up the *Commands* panel which lets you type in a command; Geomview executes the command when you hit the `Enter` key. Output from such commands is printed to standard output. Alternately, you can invoke Geomview as `geomview -c -` which causes it to read GCL commands from standard input.

GCL functions return a value, and you can nest function calls in ways which use this returned value. For example

```
(f (g a b))
```

evaluates `(g a b)` and then evaluates `(f x)` where `x` is the result returned by `(g a b)`. Geomview maintains these return values internally but does not normally print them out. To print out a return value pass it to the `echo` function. For example the `geomview-version` function returns a string representing the version of Geomview that is running, and

```
(echo (geomview-version))
```

prints out this string.

Many functions simply return `t` for success or `nil` for failure; this is the case if the documentation for the function does not indicate otherwise. These are the lisp symbols for true and false, respectively. (They correspond to the C variables `Lt` and `Lnil` which you are likely to see if you look at the source code for Geomview or some of the external modules.)

In the descriptions of the commands below several references are made to "OOGL" formats. OOGL is the data description language that Geomview uses for describing geometry, cameras, appearances, and other basic objects. For details of the OOGL formats, See Chapter 4 [OOGL File Formats], page 49. (Or equivalently, see the oogl(5) manual page, distributed with Geomview in the file /share/man/man5/oogl.5gv.

The GCL commands and argument types are listed below. Most of the documentation in this section of the manual is available within Geomview via the `?` and `??` commands. The command `(? command)` causes Geomview to print out a one-line summary of the syntax of *command*, and `(?? command)` prints out an explanation of what *command* does. You can include the wild-card character `*` in *command* to print information for a group of commands matching a pattern. For example, `(?? *emodule*)` will print all information about all commands containing the string `emodule`. `(? *)` will print a short list of all commands.

7.1 Conventions Used In Describing Argument Types

The following symbols are used to describe argument types in the documentation for GCL functions.

appearance
: is an OOGL appearance specification.

cam-id is an *id* that refers to a camera.

camera is an OOGL camera specification.

geom-id is an *id* that refers to a geometry.

geometry is an OOGL geometry specification.

id is a string which names a geometry or camera. Besides those you create, valid ones are:

`World, world, worldgeom, g0`
: the collection of all geom's

`target` selected target object (cam or geom)

`center` selected center-of-motion object

`targetcam`
: last selected target camera

`targetgeom`
: last selected target geom

`focus` camera where cursor is (or most recently was)

`allgeoms` all geom objects

`allcams` all cameras

`default, defaultcam, prototype`
: future cameras inherit default's settings

The following *id*s are used to name coordinate systems, e.g. in `pick` and `write` commands:

`World, world, worldgeom, g0`
: the world, within which all other geoms live.

`universe` the universe, in which the World, lights and cameras live. Cameras' world2cam transforms might better be called universe2cam, etc.

`self` "this Geomview object". Transform from an object to `self` is the identity; writing its geometry gives the object itself with no enclosing transform; picked points appear in the object's coordinates.

`primitive`
: (for `pick` only) Picked points appear in the coordinate system of the lowest-level OOGL primitive.

A name is also an acceptable id. Given names are made unique by appending numbers if necessary (i.e. `foo<2>`). Every geom is also named g[n] and every camera is also named c[n] (`g0` is always the worldgeom): this name is used as a prefix to keyboard commands and can also be used as a GCL id. Numbers are reused after an object is deleted. Both names are shown in the Object browser.

statement
: represents a function call. Function calls have the form `(func arg1 arg2 ... )`, where `func` is the name of the function and `arg1`, `arg2`, ... are the arguments.

transform
: is an OOGL 4x4 transformation matrix.

ntransform
: is an OOGL (N+1)x(N+1) transformation matrix.

window
: is an OOGL window specification.

7.2 GCL Reference Guide

7.2.1 !

!
: is a synonym for `shell`. See Section 7.2.129 [shell], page 129.

7.2.2 <

`(< EXPR1 EXPR2)`
: Returns t if EXPR1 is less than EXPR2. EXPR1 and EXPR2 should be either both integers or floats, or both strings.

7.2.3 =

`(= EXPR1 EXPR2)`
: Returns t if EXPR1 is equal to EXPR2. EXPR1 and EXPR2 should be either both integers or floats, or both strings.

7.2.4 >

`(> EXPR1 EXPR2)`
: Returns t if EXPR1 is greater than EXPR2. EXPR1 and EXPR2 should be either both integers or floats, or both strings.

7.2.5 *

`(* EXPR1 EXPR2)`
: Multiplies *EXPR1* and *EXPR2* and returns the result.

7.2.6 /

`(/ EXPR1 EXPR2)`
: Divides *EXPR1* by *EXPR2* and returns the result.

7.2.7 +

`(+ EXPR1 EXPR2)`
: Adds *EXPR1* and *EXPR2* and returns the result.

7.2.8 -

`(- EXPR1 EXPR2)`
: Substracts *EXPR2* from *EXPR1* and returns the result.

7.2.9 ?

`(? [command])`

Gives one-line usage summary for `command`. Command may include `*`s as wildcards; see also Section 7.2.83 [morehelp], page 121. One-line command help; lists names only if multiple commands match. `?` is a synonym for Section 7.2.61 [help], page 117,

7.2.10 ??

`(?? command)`

`command` may include `*` wildcards. Prints more info than `(? command)`. `??` is a synonym for Section 7.2.83 [morehelp], page 121.

7.2.11 |

`|` `|` is a synonym for `emodule-run`.

7.2.12 all

`(all geometry)`

returns a list of names of all geometry objects. Use e.g. '`(echo (all geometry))`' to print such a list.

`(all camera)`

returns a list of names of all cameras.

`(all emodule defined)`

returns a list of all defined external modules.

`(all emodule running)`

returns a list of all running external modules.

7.2.13 and

`(and EXPR1 EXPR2)`

Evaluate `EXPR1` and `EXPR2` and return `t` if both return non-`nil`, otherwise return `nil`.

7.2.14 ap-override

`(ap-override [on|off])`

Selects whether appearance controls should override objects' own settings. On by default. With no arguments, returns current setting.

7.2.15 backcolor

`(backcolor CAM-ID R G B)`

Set the background color of CAM-ID; R G B are numbers between 0 and 1.

7.2.16 background-image

`(background-image CAM-ID [FILENAME])`

Use the given image as the background of camera CAM-ID (which must be a real camera, not `default` or `allcams`). Centers the image on the window area.

Works only with GL and OpenGL graphics. Use `""` for filename to remove background. With no filename argument, returns name of that window's current background image, or `""`. Any file type acceptable as a texture is allowed, e.g. .ppm.gz, .sgi, etc.

7.2.17 bbox-color

`(bbox-color GEOM-ID R G B)`

Set the bounding-box color of GEOM-ID; R G B are numbers between 0 and 1.

7.2.18 bbox-draw

`(bbox-draw GEOM-ID [yes|no])`

Say whether GEOM-ID's bounding-box should be drawn; defaults to `yes` if second argument is omitted.

7.2.19 camera

`(camera CAM-ID [CAMERA])`

Specify data for *CAM-ID*; *CAMERA* is a string giving an OOGL camera specification. If no camera *CAM-ID* exists, it is created; in this case, the second argument is optional, and if omitted, a default camera is used. See Section 7.2.91 [new-camera], page 123.

7.2.20 camera-draw

`(camera-draw CAM-ID [yes|no])`

Say whether or not cameras should be drawn in CAM-ID; `yes` if omitted.

7.2.21 camera-prop

`(camera-prop { geometry object } [projective])`

Specify the object to be shown when drawing other cameras. By default, this object is drawn with its origin at the camera, and with the camera looking toward the object's -Z axis. With the `projective` keyword, the camera's viewing projection is also applied to the object; this places the object's Z=-1 and Z=+1 at near and far clipping planes, with the viewing area -1<={X,Y}<=+1. Example: (camera-prop { < cube } projective)

7.2.22 camera-reset

`(camera-reset CAM-ID)`

Reset CAM-ID to its default value.

7.2.23 car

`(car LIST)`

returns the first element of LIST.

7.2.24 cdr

`(cdr LIST)`
: returns the list obtained by removing the first element of LIST.

7.2.25 clock

`(clock)` Returns the current time, in seconds, as shown by this stream's clock. See Section 7.2.121 [set-clock], page 128. See Section 7.2.131 [sleep-until], page 130.

7.2.26 command

`(command INFILE [OUTFILE])`
: Read commands from INFILE; send corresponding responses (e.g. anything written to filename `-`) to OUTFILE, stdout by default.

7.2.27 cons

`(cons EXPR LIST)`
: Returns the list obtained by adding *EXPR* as first element of *LIST*. Note that the second argument has to be a list.

7.2.28 copy

`(copy [ID] [name])`
: Copies an object or camera. If ID is not specified, it is assumed to be target-geom. If name is not specified, it is assumed to be the same as the name of ID.

7.2.29 cursor-still

`(cursor-still [INT])`
: Sets the number of microseconds for which the cursor must not move to register as holding still. If INT is not specified, the value will be reset to the default.

7.2.30 cursor-twitch

`(cursor-twitch [INT])`
: Sets the distance which the cursor must not move (in x or y) to register as holding still. If INT is not specified, the value will be reset to the default.

7.2.31 defun

`(defun NAME (ARG1 ...) [DOCSTRING] EXPR1 ...)`
: Define a named lambda-expression, that is: define *NAME* to evaluate to the lambda-expression `(lambda (ARG1 ...) (EXPR1 ...))` when called as a function. Also, install *DOCSTRING* as response to the commands `(help NAME)` and `(morehelp NAME)`. Note that *DOCSTRING* need not contain the command-synopsis, it is generated automatically. *EXPR1* cannot be a string if `DOCSTRING` is omitted; it would be interpreted as the doc-string. The return value of `(defun ...)` is the function name. Functions can be recursive and self-modifying. It is possible to redefine builtin-functions, in this case the old definition is still

available under the name `-builtin-OLDNAME-`. Argument values may be altered by `setq`; the new binding is discarded after evaluation of the surrounding `defun`-body. The special keywords `&optional` and `&rest` have the same meaning as for anonymous lambda-expression, see there. See Section 7.2.68 [lambda], page 118. See Section 7.2.127 [setq], page 129. See Section 7.2.69 [let], page 119.

7.2.32 delete

`(delete ID)`

Delete object or camera ID.

7.2.33 dice

`(dice GEOM-ID N)`

Dice any Bezier patches within *GEOM-ID* into NxN meshes; default 10. See also the appearance attribute Section 4.1.10 [Appearances], page 53, which makes this command obsolete.

7.2.34 dimension

`(dimension [N])`

Sets or reads the space dimension for N-dimensional viewing. (Since calculations are done using homogeneous coordinates, this means matrices are (N+1)x(N+1).) With no arguments, returns the current dimension, or 0 if N-dimensional viewing has not been enabled.

7.2.35 dither

`(dither CAM-ID {on|off|toggle})`

Turn dithering on or off in that camera.

7.2.36 draw

`(draw CAM-ID)`

Draw the view in CAM-ID, if it needs redrawing. See Section 7.2.113 [redraw], page 127.

7.2.37 dump-handles

`(dump-handles)`

Dump the list of currently active handles to stdout. This function is intended for internal debugging use only.

7.2.38 echo

`(echo ...)`

Write the given data to the special file `-`. Strings are written literally; lisp expressions are evaluated and their values written. If received from an external program, `echo` sends to the program's input. Otherwise writes to geomview's own standard output (typically the terminal).

7.2.39 emodule-clear

`(emodule-clear)`

Clears the geomview application (external module) browser.

7.2.40 emodule-define

`(emodule-define NAME SHELL-COMMAND ...)`

Define an external module called NAME, which then appears in the external-module browser. The SHELL-COMMAND string is a UNIX shell command which invokes the module. See Section 7.2.44 [emodule-run], page 114, for discussion of external modules.

7.2.41 emodule-defined

`(emodule-defined modulename)`

If the given external-module name is known, returns the name of the program invoked when it's run as a quoted string; otherwise returns nil. `(echo (emodule-defined name))` prints the string.

7.2.42 emodule-isrunning

`(emodule-isrunning NAME)`

Returns Lt if the emodule NAME is running, or Lnil if it is not running. NAME is searched for in the names as they appear in the browser and in the shell commands used to execute the external modules (not including arguments).

7.2.43 emodule-path

`(emodule-path)`

Returns the current search path for external modules. Note: to actually see the value returned by this function you should wrap it in a call to echo: `(echo (emodule-path))`. See Section 7.2.123 [set-emodule-path], page 128.

7.2.44 emodule-run

`(emodule-run SHELL-COMMAND ARGS...)`

Runs the given SHELL-COMMAND (a string containing a UNIX shell command) as an external module. The module's standard output is taken as geomview commands; responses (written to filename `-`) are sent to the module's standard input. The shell command is interpreted by /bin/sh, so e.g. I/O redirection may be used; a program which prompts the user for input from the terminal could be run with:

```
(emodule-run  yourprogram  <&2)
```

If not already set, the environment variable $MACHTYPE is set to the name of the machine type. Input and output connections to geomview are dropped when the shell command terminates. Clicking on a running program's module-browser entry sends the signal SIGHUP to the program. For this to work, programs should avoid running in the background; those using FORMS or GL should call foreground() before the first FORMS or winopen() call. See

Section 7.2.40 [emodule-define], page 114. See Section 7.2.46 [emodule-start], page 115.

7.2.45 emodule-sort

`(emodule-sort)`

Sorts the modules in the application browser alphabetically.

7.2.46 emodule-start

`(emodule-start NAME)`

Starts the external module NAME, defined by emodule-define. Equivalent to clicking on the corresponding module-browser entry.

7.2.47 emodule-transmit

`(emodule-transmit NAME LIST)`

Places LIST into external module NAME's standard input. NAME is searched for in the names of the modules as they appear in the External Modules browser and then in the shell commands used to execute the external modules. Does nothing if the module NAME is not running.

7.2.48 escale

`(escale GEOM-ID FACTOR)`

Same as scale but multiplies by exp(scale). Obsolete.

7.2.49 eval

`(eval EXPR)`

Evaluate a lisp expression. If *EXPR* is an unevaluated S-expression as returned by the `(quote ...)` command then the effect will be as if calling the un-quoted expression directly. It is also possible to evaluate S-expression constructed via `car`, `cdr` and `cons`. See Section 7.2.23 [car], page 111. See Section 7.2.24 [cdr], page 112. See Section 7.2.27 [cons], page 112.

7.2.50 event-keys

`(event-keys {on|off})`

Turn keyboard events on or off to enable/disable keyboard shortcuts.

7.2.51 event-mode

`(event-mode MODESTRING)`

Set the mouse event (motion) mode; MODESTRING should be one of the following strings:

1. `"[r] Rotate"`
2. `"[t] Translate"`
3. `"[z] Cam Zoom"`
4. `"[s] Geom Scale"`

5. `"[f] Cam Fly"`
6. `"[o] Cam Orbit"`
7. `"[le] Edit Lights"`

7.2.52 event-pick

`(event-pick {on|off})`
Turn picking on or off.

7.2.53 evert

`(evert GEOM-ID [yes|no])`
Set the normal eversion state of GEOM-ID. If the second argument is omitted, toggle the eversion state.

7.2.54 exit

`(exit)` Terminates geomview.

7.2.55 ezoom

`(ezoom GEOM-ID FACTOR)`
Same as zoom but multiplies by exp(zoom). Obsolete.

7.2.56 freeze

`(freeze CAM-ID)`
Freeze CAM-ID; drawing in this camera's window is turned off until it is explicitly redrawn with `(redraw CAM-ID)`, after which time drawing resumes as normal.

7.2.57 geometry

`(geometry GEOM-ID [GEOMETRY])`
Specify the geometry for GEOM-ID. GEOMETRY is a string giving an OOGL geometry specification. If no object called GEOM-ID exists, it is created; in this case the GEOMETRY argument is optional, and if omitted, the new object GEOM-ID is given an empty geometry.

7.2.58 geomview-version

`(geomview-version)`
Returns a string representing the version of geomview that is running.

7.2.59 hdefine

`(hdefine geometry|camera|window|appearance|image|transform|ntransform name value)`
Sets the value of a handle of a given type.

```
(hdefine  <type>  <name>  <value>)
```

is generally equivalent to

```
(read <type>  { define <name> <value> })
```

except that the assignment is done when hdefine is executed, (possibly not at all if inside a conditional statement), while the `read ... define` performs assignment as soon as the text is read. See Section 4.1.9 [References], page 52. See Section 7.2.111 [read], page 126. See Section 7.2.60 [hdelete], page 117.

7.2.60 hdelete

`(hdelete [geometry|camera|window|appearance|image|transform|ntransform] name)`

Deletes the given handle. Note that the handle will not actually be deleted in case there are still other objects referring to the handle, but once those objects are gone, the handle will also automatically go away. The object the handle referes to (if any) will only be deleted if there are no other references to that object.

If the optional first argument is omitted, then the first handle matching *name* will be deleted, regardless of the type of the object it is attached to. It is not an error to call this function with a non-existent handle, but it is an error to call this funcion with the name of a non-global handle, i.e. one that was not created by `(hdefine ...)` or `(read ... { define ...})`. See Section 4.1.9 [References], page 52. See Section 7.2.111 [read], page 126. See Section 7.2.59 [hdefine], page 116.

7.2.61 help

`(help [command])`

Command may include *s as wildcards; see also Section 7.2.9 [help-synonym], page 110, One-line command help; lists names only if multiple commands match.

7.2.62 hmodel

`(hmodel CAMID {virtual|projective|conformal})`

Set the model used to display geometry in this camera. See Section 7.2.134 [space], page 130.

7.2.63 hsphere-draw

`(hsphere-draw CAMID [yes|no])`

Say whether to draw a unit sphere: the sphere at infinity in hyperbolic space, and a reference sphere in Euclidean and spherical spaces. If the second argument is omitted, `yes` is assumed.

7.2.64 if

`(if TEST EXPR1 [EXPR2])`

Evaluates TEST; if TEST returns a non-nil value, returns the value of EXPR1. If TEST returns nil, returns the value of EXPR2 if EXPR2 is present, otherwise returns nil.

7.2.65 inhibit-warning

`(inhibit-warning STRING)`

Inhibit warning inhbits geomview from displaying a particular warning message determined by STRING. At present there are no warning messages that this applies to, so this command is rather useless.

7.2.66 input-translator

`(input-translator "#prefix_string" "Bourne-shell-command")`

Defines an external translation program for special input types. When asked to read a file which begins with the specified string, geomview invokes that program with standard input coming from the given file. The program is expected to emit OOGL geometric data to its standard output. In this implementation, only prefixes beginning with # are recognized. Useful as in

```
(input-translator "#VRML" "vrml2oogl")
```

7.2.67 interest

`(interest (COMMAND [args]))`

Allows you to express interest in a command. When geomview executes that command in the future it will echo it to the communication pool from which the interest command came. *COMMAND* can be any command. Args specify restrictions on the values of the arguments; if args are present in the interest command, geomview will only echo calls to the command in which the arguments match those given in the interest command. Two special argument values may appear in the argument list. `*` matches any value. `nil` matches any value but supresses the reporting of that value; its value is reported as `nil`.

The purpose of the interest command is to allow external modules to find out about things happening inside geomview. For example, a module interested in knowing when a geom called `foo` is deleted could say `(interest (delete foo))` and would receive the string `(delete foo)` when foo is deleted.

Picking is a special case of this. For most modules interested in pick events the command `(interest (pick world))` is sufficient. This causes geomview to send a string of the form `(pick world ...)` every time a pick event (right mouse double click). See the Section 7.2.99 [pick], page 124, command for details.

7.2.68 lambda

`(lambda (ARG1 ...) EXPR1 ... EXPRN)`

A lambda expression is like a function. To "call" a lambda expression, it has to be evoked like a function: `((lambda (arg) (+ 1 arg)) 2)`. In this example, the value of the entire expression would be 3. In general, the value of the call will be the value of *EXPRN*. The first list serves to define formal parameters. The lambda expression itself is just a list, starting with the key-word lambda, followed by several quoted lists. See Section 7.2.31 [defun], page 112. See Section 7.2.127 [setq], page 129. See Section 7.2.69 [let], page 119. Note the argument list may contain the special keywords

`&optional`
: giving values to the following identifiers is optional, their default value will be `nil`

`&rest`
: all excess arguments will be collected in a list, and that list will be assigned to the following argument, like so:

```
((lambda (&rest rest) (echo rest)) a b c d)
```

The output would be `(a b c d)`.

7.2.69 let

`(let ARGUMENTS EXPR1 ... EXPRN)`

Generate a lambda expression from *EXRP1 ... EXPRN*, with the argument bindings described by *ARGUMENTS*. *ARGUMENTS* is a list of symbols (bound to `nil` by default) or lists of the form `(ARG VALUE)` where *ARG* is a symbol and not evaluated and *VALUE* is an S-expression which is first evaluated, then its value is bound to *ARG*. The entire expression evaluates to the value of *EXPRN*, the last expression in the body of the statement. The argument list must be present, but can be empty; in the latter case the `(let () ...)` statement is equivalent to a `(progn ...)`. See Section 7.2.68 [lambda], page 118. See Section 7.2.31 [defun], page 112. See Section 7.2.127 [setq], page 129.

7.2.70 lines-closer

`(lines-closer CAM-ID DIST)`

Draw lines (including edges) closer to the camera than polygons by DIST / 10^5 of the Z-buffer range. DIST = 3.0 by default. If DIST is too small, a line lying on a surface may be dotted or invisible, depending on the viewpoint. If DIST is too large, lines may appear in front of surfaces that they actually lie behind. Good values for DIST vary with the scene, viewpoint, and distance between near and far clipping planes. This feature is a kludge, but can be helpful.

7.2.71 load

`(load filename [command|geometry|camera])`

Loads the given file into geomview. The optional second argument specifies the type of data it contains, which may be `command` (geomview commands), `geometry` (OOGL geometric data), or `camera` (OOGL camera definition). If omitted, attempts to guess about the file's contents. Loading geometric data creates a new visible object; loading a camera opens a new window; loading a command file executes those commands.

7.2.72 load-path

`(load-path)`

Returns the current search path for command, geometry, files, etc. Note: to actually see the value returned by this function you should wrap it in a call to echo: `(echo (load-path))`. See Section 7.2.124 [set-load-path], page 128.

7.2.73 look

`(look [objectID] [cameraID])`

Rotates the named camera to point toward the center of the bounding box of the named object (or the origin in hyperbolic or spherical space). In Euclidean space, moves the camera forward or backward until the object appears as large as possible while still being entirely visible. Equivalent to

```
progn (
        (look-toward [objectID] [cameraID] {center | origin})
        [(look-encompass [objectID] [cameraID])]
)
```

If objectID is not specified, it is assumed to be World. If cameraID is not specified, it is assumed to be targetcam.

7.2.74 look-encompass

`(look-encompass [objectID] [cameraID])`

Moves cameraID backwards or forwards until its field of view surrounds objectID. This routine works only in Euclidean space. If objectID is not specified, it is assumed to be the world. If cameraID is not specified, it is assumed to be the targetcam. See Section 7.2.75 [look-encompass-size], page 120.

7.2.75 look-encompass-size

`(look-encompass-size [view-fraction clip-ratio near-margin far-margin])`

Sets/returns parameters used by (look-encompass). view-fraction is the portion of the camera window filled by the object, clip-ratio is the max allowed ratio of near-to-far clipping planes. The near clipping plane is 1/near-margin times closer than the near edge of the object, and the far clipping plane is far-margin times further away. Returns the list of current values. Defaults: .75 100 0.1 4.0

7.2.76 look-recenter

`(look-recenter [objectID] [cameraID])`

Translates and rotates the camera so that it is looking in the -z direction (in objectID's coordinate system) at the center of objectID's bounding box (or the origin of the coordinate system in non-Eudlidean space). In Euclidean space, the camera is also moved as close as possible to the object while allowing the entire object to be visible. Also makes sure that the y-axes of objectID and cameraID are parallel.

7.2.77 look-toward

`(look-toward [objectID] [cameraID] [origin | center])`

Rotates the named camera to point toward the origin of the object's coordinate system, or the center of the object's bounding box (in non-Euclidean space, the origin will be used automatically). Default objectID is the world, default camera is targetcam, default location to point towards is the center of the bounding box.

7.2.78 merge

`(merge {window|camera} CAM-ID { WINDOW or CAMERA ... })`

Modify the given window or camera, changing just those properties specified in the last argument. E.g.

```
(merge camera Camera { far 20 })
```

sets Camera's far clipping plane to 20 while leaving other attributes untouched.

7.2.79 merge-ap

`(merge-ap GEOM-ID APPEARANCE)`

Merge in some appearance characteristics to GEOM-ID. Appearance parameters include surface and line color, shading style, line width, and lighting.

7.2.80 merge-base-ap

`(merge-base-ap APPEARANCE)`

`merge-base-ap` is a synonym for Section 7.2.81 [merge-baseap], page 121.

7.2.81 merge-baseap

`(merge-baseap APPEARANCE)`

Merge in some appearance characteristics to the base default appearance (applied to every geom before its own apperance). Lighting is typically included in the base appearance.

7.2.82 mod

`(mod EXPR1 EXPR2)`

Divides *EXPR1* by *EXPR2* and returns the remainder.

7.2.83 morehelp

`(morehelp command)`

`command` may include `*` wildcards. Prints more info than Section 7.2.61 [help], page 117.

7.2.84 name-object

`(name-object ID NAME)`

Assign a new NAME (a string) to ID. A number is appended if that name is in use (for example, `foo -> foo<2>`). The new name, possibly with number appended, may be used as object's id thereafter.

7.2.85 ND-axes

`(ND-axes CAMID [CLUSTERNAME [Xindex Yindex Zindex [Windex]]])`

In our model for N-D viewing (enabled by (dimension)), objects in N-space are viewed by N-dimensional *camera clusters*. Each real camera window belongs to some cluster, and shows & manipulates a 3-D axis-aligned projected subspace of the N-space seen by its cluster. Moving one camera in a cluster affects its siblings.

The ND-axes command configures all this. It specifies a camera's cluster membership, and the set of N-space axes which become the 3-D camera's X, Y, and Z axes. Axes are specified by their indices, from 1 to N for an N-dimensional space. Cluster CLUSTERNAME is implicitly created if not previously known.

In principle it is possible to map the homogeneous component of a conformal 4 point to some other index; this would be done by specifying 0 for one of `Xindex`, `Yindex` or `Zindex` and giving `Windex` some positive value. This is probably not useful because Geomview does not support non-Euclidean geometries for in higher dimensions.

To read a camera's configuration, use `(echo (ND-axes CAMID))`. The return value is an array of 4 integers, the last one should 0.

7.2.86 ND-color

`(ND-color CAMID`

[(([ID] (x1 x2 . . . xN) v r g b a v r g b a . . .) ((x1 . . . xN) v r g b a v r g b a . . .) . . .)]) Specifies a function, applied to each N-D vertex, which determines the colors of N-dimensional objects as shown in camera CAMID. Each coloring function is defined by a vector (in ID's coordinate system) [x1 . . . xN] and by a sequence of value (v)/color(r g b a) tuples, ordered by increasing v. The inner product v = P.[x] is linearly interpolated in this table to give a color. If ID is omitted, the (xi) vector is assumed in universe coordinates. The ND-color command specifies a list of such functions; each vertex is colored by their sum (so e.g. green intensity could indicate projection along one axis while red indicated another. An empty list, as in (ND-color CAMID ()), suppresses coloring. With no second argument, (ND-color CAMID) returns that camera's color-function list. Even when coloring is enabled, objects tagged with the `keepcolor` appearance attribute are shown in their natural colors.

7.2.87 ND-xform

`(ND-xform OBJID [ntransform { idim odim ... }])`

Concatenate the given ND-transform with the current ND-transform of the object (apply the ND-transform to object ID, as opposed to simply setting its ND-transform). Note that ND-transforms have their homogeneous coordinate at index 0, while 3D transform have it at index 3.

7.2.88 ND-xform-get

`(ND-xform-get ID [from-ID])`

Returns the N-D transform of the given object in the coordinate system of from-ID (default `universe`), in the sense <point-in-ID-coords> * Transform = <point-in-from-ID-coords>. Note that ND-transforms have their homogeneous coordinate at index 0, while 3D transform have it at index 3.

7.2.89 ND-xform-set

`(ND-xform-set OBJID [ntransform { idim odim ... }])`

Sets the N-D transform of the given object. In dimension N, this is an (N+1)x(N+1) matrix, so in that case idim and odim are expected to be both

equal to (N+1). Note that all cameras in a camera-cluster have the same N-D transform. Note that ND-transforms have their homogeneous coordinate at index 0, while 3D transform have it at index 3.

7.2.90 new-alien

`(new-alien name [GEOMETRY])`

Create a new alien (geom not in the world) with the given name (a string). *GEOMETRY* is a string giving an OOGL geometry specification. If *GEOMETRY* is omitted, the new alien is given an empty geometry. If an object with that name already exists, the new alien is given a unique name. The light beams that are used to move around the lights are an example of aliens. They're drawn but are not controllable the way ordinary objects are: they don't appear in the object browser and the user can't move them with the normal motion modes.

7.2.91 new-camera

`(new-camera name [CAMERA])`

Create a new camera with the given name (a string). If a camera with that name already exists, the new object is given a unique name. If *CAMERA* is omitted a default camera is used.

7.2.92 new-center

`(new-center [id])`

Stop id, then set id's transform to the identity. Default id is target. Also, if the id is a camera, calls (look-recenter World id). The main function of the call to (look-recenter) is to place the camera so that it is pointing parallel to the z axis toward the center of the world.

7.2.93 new-geometry

`(new-geometry name [GEOMETRY])`

Create a new geom with the given name (a string). *GEOMETRY* is a string giving an OOGL geometry specification. If *GEOMETRY* is omitted, the new object is given an empty geometry. If an object with that name already exists, the new object is given a unique name.

7.2.94 new-reset

`(new-reset)`

Equivalent to `(progn (new-center ALLGEOMS)(new-center ALLCAMS))`.

7.2.95 NeXT

`(NeXT)` Returns `t` if running on a NeXT, `nil` if not. A relict from ancient work-station year.

7.2.96 normalization

`(normalization GEOM-ID {each|none|all|keep})`

Set the normalization status of GEOM-ID.

`none` suppresses all normalization.

`each` normalizes the object's bounding box to fit into the unit sphere, with the center of its bounding box translated to the origin. The box is scaled such that its long diagonal, sqrt((xmax-xmin)^2 + (ymax-ymin)^2 + (zmax-zmin)^2), is 2.

`all` resembles `each`, except when an object is changing (e.g. when its geometry is being changed by an external program). Then, `each` tightly fits the bounding box around the object whenever it changes and normalizes accordingly, while `all` normalizes the union of all variants of the object and normalizes accordingly.

`keep` leaves the current normalization transform unchanged when the object changes. It may be useful to apply `each` or `all` normalization apply to the first version of a changing object to bring it in view, then switch to `keep`.

7.2.97 not

`(not EXPR)`

Evaluates `EXPR`; if `EXPR` returns a non-`nil` value, returns nil, if `EXPR` returns `nil`, return `t`.

7.2.98 or

`(or EXPR1 EXPR2)`

Evaluates `EXPR1`; if `EXPR1` returns non-`nil`, return its value, if `EXPR1` returns `nil`, evaluate `EXPR2` and return its value.

7.2.99 pick

`(pick COORDSYS GEOMID G V E F P VI EI FI)`

The pick command is executed internally in response to pick events (right mouse double click).

COORDSYS
= coordinate system in which coordinates of the following arguments are specified. This can be:

`world` world coord sys

`self` coord sys of the picked geom (*GEOMID*)

`primitive`
coord sys of the actual primitive within the picked geom where the pick occurred.

GEOMID = id of picked geom

G = picked point (actual intersection of pick ray with object)

V = picked vertex, if any

E = picked edge, if any

F = picked face

P = path to picked primitive [0 or more]

VI = index of picked vertex in primitive

EI = list of indices of endpoints of picked edge, if any

FI = index of picked face

External modules can find out about pick events by registering interest in calls to `pick` via the `interest` command.

In the ND-viewing context the co-ordinates are actually ND-points. They correspond to the 3D points of the pick relative to the sub-space defined by the viewport of the camera where the pick occurred. The co-ordinates are then padded with zeroes and transformed back to the co-ordinate system defined by *COORDSYS*.

7.2.100 pick-invisible

`(pick-invisible [yes|no])`

Selects whether picks should be sensitive to objects whose appearance makes them invisible; default yes. With no arguments, returns current status.

7.2.101 pickable

`(pickable GEOM-ID {yes|no})`

Say whether or not GEOM-ID is included in the pool of objects that could be returned from the pick command.

7.2.102 position

`(position objectID otherID)`

Set the transform of objectID to that of otherID.

7.2.103 position-at

`(position-at objectID otherID [center | origin])`

Translate objectID to the center of the bounding box or the origin of the coordinate system of otherID (parallel translation). Default is center.

7.2.104 position-toward

`(position-toward objectID otherID [center | origin])`

Rotate objectID so that the center of the bounding box or the origin of the coordinate system of the otherID lies on the positive z-axis of the first object. Default is the center of the bounding box.

7.2.105 process-events

`(process-events)`

Pass control back to the event loop of Geomview, then continue evaluating the current command-script. If the current input stream has been put to sleep by

one of the `(sleep-...)` commands, then the control remains by the main-loop until the current input streams "sleep" is over. See Section 7.2.131 [sleep-until], page 130. See Section 7.2.130 [sleep-for], page 129.

7.2.106 progn

`(progn STATEMENT [ ... ])`

evaluates each STATEMENT in order and returns the value of the last one. Use progn to group a collection of commands together, forcing them to be treated as a single command.

7.2.107 quit

`(quit)` `quit` is a synonym for Section 7.2.54 [exit], page 116.

7.2.108 quote

`(quote EXPR)`

returns the symbolic lisp expression *EXPR* without evaluating it. Note, however, that `quote` parses *EXPR* as if it should be evaluated. See Section 7.2.49 [eval], page 115.

7.2.109 rawevent

`(rawevent dev val x y t)`

Enter the specified raw event into the event queue. The arguments directly specify the members of the event structure used internally by geomview. This is the lowest level event handler and is not intended for general use.

7.2.110 rawpick

`(rawpick CAMID X Y)`

Process a pick event in camera CAMID at location (X,Y) given in integer pixel coordinates. This is a low-level procedure not intended for external use.

7.2.111 read

`(read {geometry|camera|appearance|image|ntransform|transform|command}`
`{GEOMETRY or CAMERA or ...})`

Read and interpret the text in ... as containing the given type of data. Useful for defining objects using OOGL reference syntax, e.g.

```
(geometry  thing { INST  transform : T    geom : fred })
(read  geometry  { define fred
   QUAD
   1 0 0  0 1 0  0 0 1  1 0 0
})
(read  transform { define T <myfile})
```

See Section 4.1.9 [References], page 52. See Section 7.2.59 [hdefine], page 116. See Section 7.2.60 [hdelete], page 117.

7.2.112 real-id

`(real-id ID)`

Returns a string canonically identifying the given ID, or `nil` if the object does not exist. Examples: (if (real-id fred) (delete fred)) deletes `fred` if it exists but reports no error if it doesn't, and (if (= (real-id targetgeom) (real-id World)) () (delete targetgeom)) deletes `targetgeom` if it is different from the World.

7.2.113 redraw

`(redraw CAM-ID)`

States that the view in CAM-ID should be redrawn on the next pass through the main loop or the next invocation of `draw`.

7.2.114 regtable

`(regtable)`

shows the registry table

7.2.115 rehash-emodule-path

`(rehash-emodule-path)`

Rebuilds the application (external module) browser by reading all .geomview-* files in all directories on the emodule-path. Primarily intended for internal use; any applications defined by `(emodule-define ...)` commands outside of the .geomview-* files on the emodule-path will be lost. Does not sort the entries in the brower. See Section 7.2.45 [emodule-sort], page 115. See Section 7.2.40 [emodule-define], page 114.

7.2.116 replace-geometry

`(replace-geometry GEOM-ID PART-SPECIFICATION GEOMETRY)`

Replace a part of the geometry for GEOM-ID.

7.2.117 rib-display

`(rib-display [frame|tiff] FILEPREFIX)`

Set Renderman display to framebuffer (popup screen window) or a TIFF format disk file. FILEPREFIX is used to construct names of the form `prefixNNNN.suffix`. (i.e. foo0000.rib) The number is incremented on every call to `rib-snapshot` and reset to 0000 when `rib-display` is called. TIFF files are given the same prefix and number as the RIB file (i.e. foo0004.rib generates foo0004.tiff). The default FILEPREFIX is `geom` and the default format is TIFF. (Note that geomview just generates a RIB file, which must then be rendered.)

7.2.118 rib-snapshot

`(rib-snapshot CAM-ID [filename])`

Write Renderman snapshot (in RIB format) of CAM-ID to <filename>. If no filename specified, see Section 7.2.117 [rib-display], page 127, for an explanation of the filename used.

7.2.119 scale

`(scale GEOM-ID FACTOR [FACTORY FACTORZ])`

Scale GEOM-ID, multiplying its size by FACTOR. The factors should be positive numbers. If FACTORY and FACTORZ are present and non-zero, the object is scaled by FACTOR in x, by FACTORY in y, and by FACTORZ in z. If only FACTOR is present, the object is scaled by FACTOR in x, y, and z. Scaling only really makes sense in Euclidean space. Mouse-driven scaling in other spaces is not allowed; the scale command may be issued in other spaces but should be used with caution because it may cause the data to extend beyond the limits of the space.

7.2.120 scene

`(scene CAM-ID [GEOMETRY])`

Make CAM-ID look at GEOMETRY instead of at the universe.

7.2.121 set-clock

`(set-clock TIME)`

Adjusts the clock for this command stream to read *TIME* (in seconds) as of the moment the command is received. See Section 7.2.131 [sleep-until], page 130. See Section 7.2.25 [clock], page 112.

7.2.122 set-conformal-refine

`(set-conformal-refine CMX [N [SHOWEDGES]])`

Sets the parameters for the refinement algorithm used in drawing in the conformal model. CMX is the cosine of the maximum angle an edge can bend before it is refined. Its value should be between -1 and 1; the default is 0.95; decreasing its value will cause less refinement. N is the maximum number of iterations of refining; the default is 6. SHOWEDGES, which should be `no` or `yes`, determines whether interior edges in the refinement are drawn.

7.2.123 set-emodule-path

`(set-emodule-path (PATH1 ... PATHN))`

Sets the search path for external modules. The PATHi should be pathnames of directories containing, for each module, the module's executable file and a .geomview-<modulename> file which contains an (emodule-define ...) command for that module. This command implicitly calls (rehash-emodule-path) to rebuild the application browser from the new path setting. The special directory name `+` is replaced by the existing path, so e.g. (set-emodule-path (mydir +)) prepends mydir to the path.

7.2.124 set-load-path

`(set-load-path (PATH1 ... PATHN))`

Sets search path for command, geometry, files, etc. The PATHi are strings giving the pathnames of directories to be searched. The special directory name

`+` is replaced by the existing path, so e.g. (set-load-path (mydir `+`)) prepends mydir to the path.

7.2.125 set-motionscale

`(set-motionscale X)`

Set the motion scale factor to X (default value 0.5). These commands scale their motion by an amount which depends on the distance from the frame to the center and on the size of the frame. Specifically, they scale by dist + scaleof(frame) * motionscale where dist is the distance from the center to the frame and motionscale is the motion scale factor set by this function. Scaleof(frame) measures the size of the frame object.

7.2.126 setenv

`(setenv name string)`

sets the environment variable `name` to the value *string*; the name is visible to geomview (as in pathnames containing `$name`) and to processes it creates, e.g. external modules.

7.2.127 setq

`(setq SYM EXPR)`

Bind the symbol *SYM* to the value of *EXPR*. *SYM* must be an unqualified symbol, i.e. not quoted, and literal: `(setq "foo" bar)` will not work. Likewise `(setq (bar STUFF) foo)` will also not work, even if `(bar ...)` would evaluate to an unqualified symbol: varible names must be literals. Note that calling `(setq SYM ...)` will alter the value of *SYM* within the current name-space: if *SYM*, e.g., is bound as local variable by a lambda, let or defun expression, then `(setq SYM ...)` will change the value of the local variable, the global binding will remain unchanged. It is NOT possible to un-bind a symbol. However, subsequent `(setq SYM ...)` invocations will re-bind *SYM* to another value and free the lisp-object previously bound to *SYM*. See Section 7.2.68 [lambda], page 118. See Section 7.2.31 [defun], page 112. See Section 7.2.69 [let], page 119.

7.2.128 sgi

`(sgi)` Returns `t` if running on a sgi machine, `nil` if not. A relict from the old workstation years.

7.2.129 shell

`(shell SHELL-COMMAND)`

Execute the given UNIX SHELL-COMMAND using /bin/sh. Geomview waits for it to complete and will be unresponsive until it does. A synonym is `!`.

7.2.130 sleep-for

`(sleep-for TIME)`

Suspend reading commands from this stream for TIME seconds. Commands already read will still be executed; `sleep-for` inside `progn` won't delay execution of the rest of the progn's contents.

7.2.131 sleep-until

`(sleep-until TIME)`

Suspend reading commands from this stream until TIME (in seconds). Commands already read will still be executed; `sleep-until` inside `progn` won't delay execution of the rest of the progn's contents. Time is measured according to this stream's clock, as set by `set-clock`; if never set, the first sleep-until sets it to 0 (so initially (sleep-until TIME) is the same as (sleep-for TIME)). Returns the number of seconds until TIME.

7.2.132 snapshot

`(snapshot CAM-ID FILENAME [FORMAT [XSIZE [YSIZE]]])`

Save a snapshot of *CAM-ID* in the *FILENAME* (a string). The *FORMAT* argument is optional; it may be "ppmscreen" (the default), "ps", "ppm", "sgi" (on SGI machines), "ppmosmesa" (if built with libOSMesa) or "ppmosglx". A "ppmscreen" snapshot is created by reading the image directly from the given window; the window is popped above other windows and redrawn first, then its contents are written as a PPM format image. A "ppmosmesa" snapshot is drawn by Mesa's software renderer into a memory buffer in RAM. A "ppmosglx" snapshot is rendered into a GLX Pixmap buffer, which is also off-screen but may or may not reside in video RAM. Rendering may or may not be accelerated. The problem with on-screen snapshots is that the window must be mapped and not obscured by other windows. So on-screen snapshots will not work in the background, or when a screen saver is active. With "ps", dumps a Postscript picture representing the view from that window; hidden-surface removal might be incorrect. With "ppm", dumps a PPM-format image produced by geomview's internal software renderer; this may be of arbitrary size. If the *FILENAME* argument begins with "|", it's interpreted as a `/bin/sh` command to which the PPM or PS data should be piped. Optional *XSIZE* and *YSIZE* values are relevant only for "ppm" formats, and render to a window of that size (or scaled to that size, with aspect fixed, if only *XSIZE* is given)

7.2.133 soft-shader

`(soft-shader CAM-ID {on|off|toggle})`

Select whether to use software or hardware shading in that camera.

7.2.134 space

`(space {euclidean|hyperbolic|spherical})`

Set the space associated with the world.

7.2.135 stereowin

`(stereowin CAM-ID [no|horizontal|vertical|colored] [gapsize])`

Configure CAM-ID as a stereo window. no: entire window is a single pane, stereo disabled
horizontal: split left/right: left is stereo eye#0, right is #1.

vertical: split top/bottom: bottom is eye#0, top is #1.
colored: panes overlap, red is stereo eye#0, cyan is #1.

A gap of `gapsize` pixels is left between subwindows; if omitted, subwindows are adjacent. If both layout and gapsize are omitted, e.g. (stereowin CAM-ID), returns current settings as a `(stereowin ...)` command list. This command doesn't set stereo projection; use `merge camera` or `camera` to set the stereyes transforms, and `merge window` or `window` to set the pixel aspect ratio & window position if needed.

7.2.136 time-interests

`(time-interests deltatime initial prefix [suffix])`

Indicates that all interest-related messages, when separated by at least `deltatime` seconds of real time, should be preceded by the string `prefix` and followed by `suffix`; the first message is preceded by `initial`. All three are printf format strings, whose argument is the current clock time (in seconds) on that stream. A `deltatime` of zero timestamps every message. Typical usage:
(time-interests .1 `(set-clock %g) (sleep-until %g)`) or
(time-interests .1 `(set-clock %g)` "(sleep-until %g) (progn (set-clock %g)" ")") or
(time-interests .1 "(set-clock %g)" "(if (> 0 (sleep-until %g)) (" "))".

7.2.137 transform

`(transform objectID centerID frameID`
`[rotate|translate|translate-scaled|scale] x y z [dt [smooth]])`

Apply a motion (rotation, translation, scaling) to object `objectID`; that is, construct and concatenate a transformation matrix with objectID's transform. The 3 IDs involved are the object that moves, the center of motion, and the frame of reference in which to apply the motion. The center is easiest understood for rotations: if centerID is the same as objectID then it will spin around its own axes; otherwise the moving object will orbit the center object. There is the special keyword `bbox-center` which may be used for `centerID`. As a result the motion will be relative to the center of the bounding box of `objectID`. Normally frameID, in whose coordinate system the (mouse) motions are interpreted, is `focus`, the current camera. Translations can be scaled proportional to the distance between the target and the center. Support for spherical and hyperbolic as well as Euclidean space is built-in: use the `space` command to change spaces. With type `rotate` x, y, and z are floats specifying angles in RADIANS. For types `translate` and `translate-scaled` x, y, and z are floats specifying distances in the coordinate system of the center object.

The optional `dt` field allows a simple form of animation; if present, the object moves by just that amount during approximately `dt` seconds, then stops. If present and followed by the `smooth` keyword, the motion is animated with a 3t^2-2t^3 function, so as to start and stop smoothly. If absent, the motion is applied immediately.

7.2.138 transform-incr

`(transform-incr objectID centerID frameID [rotate|translate|translate-scaled|scale] x y z [origin|bbox-center] [dt [smooth]])`

Apply continuing motion: construct a transformation matrix and concatenate it with the current transform of objectID every refresh (sets objectID's incremental transform). Same syntax as transform. If optional `dt` argument is present, the object is moved at each time step such that its average motion equals one instance of the motion per `dt` seconds. E.g. (transform-incr World World World rotate 6.28318 0 0 10.0) rotates the World about its X axis at 1 turn (2pi radians) per 10 seconds.

7.2.139 transform-set

`(transform-set objectID centerID frameID [rotate|translate|translate-scaled|scale] x y z [origin|bbox-center])`

Set objectID's transform to the constructed transform. Same syntax as transform.

7.2.140 truncate

`(truncate EXPR)`

Rounds *EXPR* towards zero.

7.2.141 ui-cam-focus

`(ui-cam-focus [focus-change|mouse-cross])`

Set the focus policy for the camera windows. The default is `mouse-cross`: a camera is made the active camera (for interactive mouse events) when the mouse cursor crosses the window. Because this means it can become complicated to activate a specific camera (in the context of multiple camera windows) there is also the option to only change the camera focus when the window-manager decides to give it the focus for input events. So, after specifying `focus-change` it depends on the focus-change configuration of your window-manager when a camera becomes the active camera for mouse-interaction. To change this behaviour permanently you could, e.g., place the following line in your `${HOME}/.geomview` file (see Chapter 5 [Customization], page 83):

```
(progn
  (ui-cam-focus focus-change)
  ... # other stuff
)
```

7.2.142 ui-center

`(ui-center ID)`

Set the center for user interface (i.e. mouse) controlled motions to object ID.

7.2.143 ui-center-origin

`(ui-center-origin [origin|bbox-center])`

Set the origin of the coordinate system of the `CENTER` object for user interface (i.e. mouse) controlled motions. The keyword `origin` means to use the origin of the coordinate system of the currently selected object, while `bbox-center` means to use the center of the bounding box of the current object. The keyword `bbox-center` makes no sense if the geometry is non-Euclidean. Using either `bbox-center` or `origin` does not make a difference if the center object is not a *geometry*, e.g. if it is a camera. Same holds if the World is the currently selected object.

7.2.144 ui-emotion-program

`(ui-emotion-program PROGRAM)`

This is an obsolete command. Use its new eqivalent Section 7.2.40 [emodule-define], page 114, instead.

7.2.145 ui-emotion-run

`(ui-emotion-run EMODULE)`

This is an obsolete command. Use its new eqivalent Section 7.2.46 [emodule-start], page 115, instead.

7.2.146 ui-freeze

`(ui-freeze {on|off})`

Toggle updating user interface panels. Off by default.

7.2.147 ui-html-browser

`(ui-html-browser BROWSER)`

Use *BROWSER* for viewing the *HTML*-version of the manual when the '`Manual (HTML)`' menu item is selected in the '`Help`'-menu. If the `(ui-html-browser ...)` command was never executed, then the default is to use the browser stored in the `WEBBROWSER` environment variable. If the environment variable is unset then the default is compile-time dependent.

7.2.148 ui-motion

`(ui-motion {inertia|constrain|own-coordinates} {on|off})`

Enable or disable certain properties of mouse-controlled motion. The purpose of this command is to give access to the respective toggles of the *Main* panel's *Motion* menu through GCL commands. See Section 3.5 [Mouse Motions], page 28.

`inertia` Normally, moving objects have inertia: if the mouse is still moving when the button is released, the selected object continues to move. When `inertia` is off, objects cease to move as soon as you release the mouse.

`constrain`

It's sometimes handy to move an object in a direction aligned with a coordinate axis: exactly horizontally or vertically. Calling

`(ui-motion constrain on)` changes the interpretation of mouse motions to allow this; approximately-horizontal or approximately-vertical mouse dragging becomes exactly horizontal or vertical motion. Note that the motion is still along the X or Y axes of the camera in which you move the mouse, not necessarily the object's own coordinate system.

`own-coordinates`
: It's sometimes handy to move objects with respect to the coordinate system where they were defined, rather than with respect to some camera's view. When `(ui-motion own-coordinates on)` has been called, all motions are interpreted that way: dragging the mouse rightward in translate mode moves the object in its own +*X* direction, and so on. May be especially useful in conjunction with the `(ui-motion constrain on)` command.

7.2.149 ui-panel

`(ui-panel PANELNAME {on|off} [WINDOW])`
: Do or don't display the given user-interface panel. Case is ignored in panel names. Current *PANELNAMEs* are:

geomview	main panel
tools	motion controls
appearance	appearance controls
cameras	camera controls
lighting	lighting controls
obscure	obscure controls (doesn't seem to exist any more)
materials	material properties controls
command	command entry box
credits	geomview credits

By default, the `geomview` and `tools` panels appear when geomview starts. If the optional Window is supplied, a `position` clause (e.g. (ui-panel obscure on { position xmin xmax ymin ymax }) sets the panel's default position. (Only xmin and ymin values are actually used.) A present but empty Window, e.g. `(ui-panel obscure on {)}` causes interactive positioning.

7.2.150 ui-pdf-viewer

`(ui-pdf-viewer VIEWER)`
: Use the executable *VIEWER* for viewing the *PDF*-version of the manual when the '`Manual (PDF)`' menu-item is selected in the '`Help`'-menu. If the `(ui-pdf-viewer ...)` command was never executed, then the default is to use the browser stored in the `PDFVIEWER` environment variable. If the environment variable is unset then the default is compile-time dependent.

7.2.151 ui-target

`(ui-target ID [yes|no])`

Set the target of user actions (the selected line of the target object browser) to ID. The second argument specifies whether to make ID the current object regardless of its type. If `no`, then ID becomes the current object of its type (geom or camera). The default is `yes`. This command may result in a change of motion modes based on target choice.

7.2.152 uninterest

`(uninterest (COMMAND [args]))`

Undoes the effect of an `interest` command. (COMMAND [args]) must be identical to those used in the Section 7.2.67 [interest], page 118, command.

7.2.153 update

`(update [timestep_in_seconds])`

Apply each incremental motion once. Uses timestep if it's present and nonzero; otherwise motions are proportional to elapsed real time.

7.2.154 update-draw

`(update-draw CAM-ID [timestep_in_seconds])`

Apply each incremental motion once and then draw CAM-ID. Applies `timestep` seconds' worth of motion, or uses elapsed real time if `timestep` is absent or zero.

7.2.155 while

`(while TEST BODY)`

Iterate: *evaluate TEST, if non nil, evaluate BODY*.

7.2.156 window

`(window CAM-ID WINDOW)`

Specify attributes for the window of CAM-ID, e.g. its size or initial position, in the OOGL Window syntax. The special CAM-ID `default` specifies properties of future windows (created by Section 7.2.19 [camera], page 111, or Section 7.2.91 [new-camera], page 123).

7.2.157 winenter

`(winenter CAM-ID)`

Tell geomview that the mouse cursor is in the window of CAM-ID. This function is for development purposes and is not intended for general use.

7.2.158 write

`(write {command|geometry|camera|transform|ntransform|window|bbox} FILENAME [ID|(ID ...)] [self|world|universe|otherID])`

write description of ID in given format to FILENAME. Last parameter chooses coordinate system for geometry & transform: self: just the object, no transformation or appearance (geometry only) world: the object as positioned within

the World. universe: object's position in universal coordinates; includes World-transform other ID: the object transformed to otherID's coordinate system.

A filename of `-` is a special case: data are written to the stream from which the 'write' command was read. For external modules, the data are sent to the module's standard input. For commands not read from an external program, `-` means geomview's standard output (see Section 7.2.26 [command], page 112).

The ID can either be a single id or a parenthesized list of ids, like `g0` or `(g2 g1 dodec.off)`.

7.2.159 write-comments

`(write-comments FILENAME GEOMID PICKPATH)`

write OOGL COMMENT objects in the GEOMID hierarchy at the level of the pick path to FILENAME. Specifically, COMMENTS at level (a b c ... f g) will match pick paths of the form (a b c ... f *) where * includes any value of g, and also any values of possible further indices h,i,j, etc. The pick path (returned in the `pick` command) is a list of integer counters specifying a subpart of a hierarchical OOGL object. Descent into a complex object (LIST or INST) adds a new integer to the path. Traversal of simple objects increments the counter at the current level. Individual COMMENTS are enclosed by curly braces, and the entire string of zero, one, or more COMMENTS (written in the order in which they are encountered during hierarchy traversal) is enclosed by parentheses.

Note that arbitrary data can only be passed through the OOGL libraries as full-fledged OOGL COMMENT objects, which can be attached to other OOGL objects via the LIST type as described above. Ordinary comments in OOGL files (i.e. everything after '#' on a line) are ignored at when the file is loaded and cannot be returned.

7.2.160 write-handle

`(write-handle PREFIX FILENAME HANDLE)`

Writes the object underlying the given handle to *FILENAME*. This function is intended for internal debugging use only.

7.2.161 write-sexpr

`(write-sexpr FILENAME LISPOBJECT)`

Writes the given LISPOBJECT to FILENAME. This function is intended for internal debugging use only.

7.2.162 xform

`(xform ID TRANSFORM)`

Concatenate TRANSFORM with the current transform of the object (apply TRANSFORM to object ID).

7.2.163 xform-incr

`(xform-incr ID TRANSFORM)`

Apply continual motion: concatenate TRANSFORM with the current transform of the object every refresh (set object ID's incremental transform to TRANSFORM).

7.2.164 xform-set

`(xform-set ID TRANSFORM)`

Overwrite the current object transform with TRANSFORM (set object ID's transform to TRANSFORM).

7.2.165 zoom

`(zoom CAM-ID FACTOR)`

Zoom CAM-ID, multiplying its field of view by FACTOR. FACTOR should be a positive number.

8 Non-Euclidean Geometry

Geomview supports hyperbolic and spherical geometry as well as Euclidean geometry. The three buttons at the bottom of the *Main* panel are for setting the current geometry type.

In each of the three geometries, three models are supported: *Virtual*, *Projective*, and *Conformal*. You can change the current model with the *Model* browser on the *Camera* panel. Each Geomview camera has its own model setting.

The default model is all three spaces is *Virtual*. This corresponds to the camera being in the same space as, and moving under the same set of transformations as, the geometry itself.

In Euclidean space *Virtual* is the most useful model. The other models were implemented for hyperbolic and spherical spaces and just happen to work in Eucldiean space as well: *Projective* is the same as *Virtual* but by default displays the unit sphere, and *Conformal* displays everything inverted in the unit sphere.

In hyperbolic space, the *Projective* model setting gives a view of the projective ball model of hyperbolic 3-space imbedded in Euclidean space. The camera is initially outside the unit ball. In this model, the camera moves by Euclidean motions and geometry moves by hyperbolic motions. *Conformal* model is similar but shows the conformal ball model instead.

In spherical space, the *Projective* model gives a view of half of the 3-sphere imbedded in Euclidean 3-space. Spherical motions give rise to projective transformations in the *Projective* model, and to Möbius transformations in the *Conformal* model. In both of these models the camera moves by Euclidean motions.

In *Projective* and *Conformal* models, the unit sphere is drawn by default. You can turn it off and on at will using the *Draw Sphere* button in the *Camera* panel. In the *Conformal* model, polygons and edges are subdivided as necessary to make them look curved. The parameters which determine this subdivision can be set with the `set-conformal-refine` GCL command. See Section 7.2.122 [set-conformal-refine], page 128.

There are several sample hyperbolic space objects in the `data/geom/hyperbolic` subdirectory of the Geomview directory. Likewise, the subdirectory `data/geom/spherical` contains several sample spherical space objects.

9 Mathematica Graphics in Geomview or RenderMan

Geomview comes with some Mathematica packages that let you use Geomview to display Mathematica graphics. Mathematica is a commercial mathematical software system available from Wolfram Research, Inc.

There are two ways to do this.

1. Use Mathematica to write a graphics object to a file in OOGL format or in RIB format.
2. Use Geomview as the default display for all 3D graphics output in Mathematica.

You can also use these packages to save Mathematica graphics in RenderMan (RIB) format.

Since the format of Mathematica graphics objects is different from the OOGL formats, both of these methods involve translating Mathematica graphics to OOGL format. Geomview is distributed with a Mathematica package which does this translation. Before doing either of the above you must install this package.

9.1 Using Mathematica to generate OOGL files

The package `OOGL.m` allows Mathematica to write graphics objects in OOGL format. To use it, give the command `<< OOGL.m` to Mathematica to load the package. The `WriteOOGL[`*file*`,`*graphics*`]` command writes an OOGL description of the 3D graphics object *graphics* to the file named *file*.

This package also provides the `Geomview` command which sends a 3D graphics object to Geomview. The first time you use this command it starts up a copy of Geomview. Later calls send the graphics to the same Geomview. There are two ways to use the `Geomview` command.

`Geomview[`*graphics*`]`
: Sends the 3D graphics object *graphics* to Geomview as a geom named `Mathematica`. Subsequent usage of `Geomview[`*graphics*`]` replaces the `Mathematica` object in Geomview with the new *graphics*.

`Geomview[`*name*`,`*graphics*`]`
: Sends the 3D graphics object *graphics* to Geomview as a geom named *name*. You can use multiple calls of this form with different names to cause Geomview to display several Mathematica objects at once and allow independent control over them.

```
% math
Mathematica 2.0 for SGI Iris
Copyright 1988-91 Wolfram Research, Inc.
 -- GL graphics initialized --

In[1] := <<OOGL.m

In[2] := Plot3D[Sin[x + Sin[y]], {x,-2,2},{y,-2,2}]

Out[2] := -Graphics3D-
```

This displays graphics in the usual Mathematica way here.

```
In[3] := WriteOOGL["math.oogl", %2]

Out[3] := -Graphics3D-
```

This displays nothing new but writes the file `math.oogl`. You can now load that file into Geomview on any computer. Alternately, you can use the `Geomview` command to start up a copy of Geomview from within Mathematica.

```
In[5] := Geomview[%2]

Out[5] := -Graphics3D-
```

9.2 Using Geomview as Mathematica's Default 3D Display

The package `Geomview.m` arranges for Geomview to be the default display program for 3D graphics in Mathematica. To load it, give the command `<< Geomview.m` to Mathematica. Thereafter, whenever you display 3D graphics with `Plot3D` or `Show`, Mathematica will send the graphics to Geomview.

Loading `Geomview.m` implicitly loads `OOGL.m` as well, so you can use the `Geomview` and `WriteOOGL` as described above after loading `Geomview.m`. You do not have to separately load `OOGL.m`.

```
% math
Mathematica 2.0 for SGI Iris
Copyright 1988-91 Wolfram Research, Inc.
 -- GL graphics initialized --

In[1] := <<Geomview.m

In[2] := Plot3D[x^2 + y^2, {x, -2, 2}, {y, -2, 2}]

Out[2] := -SurfaceGraphics-
```

This invokes geomivew and loads the graphics object into it.

```
In[3] := Plot3D[{x*y + 6, RGBColor[0,x,y]}, {x,0,1}, {y,0,1}]

Out[3] := -SurfaceGraphics-
```

This replaces the previous Geomview object by the new object.

```
In[4] := Geomview[{%2,%3}]

Out[4] := {-SurfaceGraphics-, -SurfaceGraphics-}
```

This displays both objects at once. You also can have more than one Mathematica object at a time on display in Geomview, and have separate control over them, by using the `Geomview` command with a name, See Section 9.1 [OOGL.m], page 139.

```
In[5] := Graphics3D[ {RGBColor[1,0,0], Line[{ {2,2,2},{1,1,1} }] }]

Out[5] := -Graphics3D-
```

```
In[6] := Geomview["myline", %5]
```

This addes the `Line` specified in `In[5]` to the existing Geomview display. It can be controlled independently of the "Mathematica" object, which is currently the list of two plots.

```
In[7] := <<GL.m
```

If you're on an SGI, loading `GL.m` returns Mathematica to its usual 3D graphics display. The following plot will appear in a normal static Mathematica window.

```
In[8] := ParametricPlot3D[{Sin[x],Sin[y],Sin[x]*Cos[y]}, {x,0,Pi},{y,0,Pi}]

Out[8] := -Graphics3D-
```

We can return to Geomview graphics at any time by reloading `Geomview.m`.

```
In[9] := <<Geomview.m

In[10] := Show[%8]

Out[10] := -Graphics3D-

In[11] := ParametricPlot3D[
           {(2*(Cos[u] + u*Sin[u])*Sin[v])/(1 + u^2*Sin[v]^2),
           (2*(Sin[u] - u*Cos[u])*Sin[v])/(1 + u^2*Sin[v]^2),
           Log[Tan[v/2]] + (2*Cos[v])/(1 + u^2*Sin[v]^2)},
          {u,-4,4},{v,.01,Pi-.01}]

Out[11] := -Graphics3D-
```

This last plot is Kuen's surface, a surface of constant negative curvature. Parametrization from Alfred Gray's *Modern Differential Geometry of Curves and Surfaces* textbook.

9.3 Using Mathematica to generate RenderMan files

In addition to the `WriteOOGL` and `Geomview` commands described above, the package `OOGL.m` also defines the command `WriteRIb` which writes a 3D graphics object to a RenderMan RIB file: `WriteRIB[`*`file`*`,` *`graphics`*`]` writes *graphics* to file *file*. RenderMan is a commercial rendering system available from Pixar, Inc., which can produce extremely high quality images.

```
In[1] := <<OOGL.m

In[2] := <<Graphics/Polyhedra.m

In[3] := Graphics3D[Cube[]]

Out[3] := -Graphics3D-

In[4] := WriteRIB["cube.rib", %3]

Out[4] := -Graphics3D-
```

This generates the file `math.rib`. This is a ready-to-render RIB file of the given geometry, using a default camera position, lighting, and the "plastic" shader. In a shell window,

type `render cube.rib` to generate the image file `mma.tiff`. Of course, you need to have RenderMan installed for this to work. A shortcut to render from inside Mathematica is `WriteRIB["!render", foo]`.

`WriteRIb` works by first converting the Mathematica graphics object to OOGL format using `WriteOOGL` and then calls an external program `oogl2rib` to convert OOGL to RIB format. The oogl2rib program takes several options which you can specify in a string as an optional third argument to `WriteRIb`. The default option string is `" -n mma.tiff "`, which indicates that the RIB file should generate a rendered TIFF file named `mma.tiff`. A particularly useful option is `-g`, which tells oogl2rib to convert only the geometry into a RIB fragment. You can insert that fragment into a full RIB file of your own making with camera positions and shaders of your choice, to harness the full power of RenderMan.

The full usage of oogl2rib is:

`oogl2rib [-n` *name*`] [-B` *r*`,`*g*`,b] [-w` *width*`] [-h` *height*`] [-fgb] [`*infile*`] [`*outfile*`]`

By default it reads from stdin and writes to stdout. Either *infile* or *outfile* may be `-`, which means use stdin/stdout. The options are:

`-n` *name* — Use *name* for the name of the rendered TIFF file (default "geom.tiff") or framebuffer window (default "geom.rib").

`-B` *r*`,`*g*`,b` — Use background color (*r*,*g*,b). Each component ranges from 0 to 1. Default: none.

`-w` *width* `-h` *height* — Rendered frame will be *width* by *height* pixels.

`-f` — RIB file renders to on-screen framebuffer instead of TIFF file.

`-g` — Output only the geometry in RIB format.

`-b` — Output only a Quick Renderman clip object. Ignores -nBwhf.

9.4 Using Geomview and Mathematica on Different Computers

It is possible to use Geomview to display graphics generated by Mathematica running on a different computer. If you want to use Mathematica on a computer that is not networked with your Geomview computer, you can write out *chunk* files in Mathematica which you transfer to the Geomview computer and then translate to OOGL format for displaying in Geomview.

9.4.1 Using a Networked Geomview Host

The `Geomview` command looks at the `DISPLAY` or `REMOTEHOST` environment variables to try to determine if you are logged in from another computer. If either of these indicates that you are, `Geomview` will attempt to run Geomview on that computer. In order for this to work, your network must be configured such that the Mathematica computer can successfully `rsh` to the Geomview computer without giving a password.

You can also explicitly set the `DisplayHost` option to the `Geomview` command to a string which is the desired hostname, for example:

```
In[1] := << OOGL.m
```

```
In[2] := Plot3D[Sin[x + Sin[y]], {x,-2,2},{y,-2,2}]

Out[2] := -Graphics3D-

In[3] := Geomview[%3, DisplayHost->"riemann"]
```

This displays the graphics `%3` on the remote host named `riemann`.

`Geomview` recognizes the string `"local"` as a value for `$DisplayHost`; it forces the graphics to be displayed on the local machine.

In addition to knowing the name of the machine you want to run Geomview on, `Geomview` needs to know the type of that machine (the setting of the CPU variable that corresponds to the machine; see Section 10.2 [Source Code Installation], page 148). By default, `Geomview` assumes that it is the same kind of computer as the one you are running Mathematica on. The `MachType` option lets you explicitly specify the type of the `DisplayHost` computer; it should be one of the strings `"sgi"` or `"next"` or `"x11"`.

You can use `SetOptions` to change the default `DisplayHost` and `MachType`. For example,

```
In[4] := SetOptions[Geomview, DisplayHost->"riemann", MachType->"sgi"]
```

arranges for `Geomview` to run Geomview on an SGI workstation named `riemann`.

9.4.2 Transporting Mathematica Files to Geomview by Hand

The auxilliary function `WriteChunk` is for those who can only use Mathematica on a computer that Geomview isn't installed on. `WriteChunk[`*`file`*`, `*`graphics`*`]` generates a file named *file* which contains the graphics object *graphics* in the format accepted by `math2oogl`.

You can transfer that file to a computer that has Geomview installed on it and then use the programs `math2oogl`, `oogl2rib`, and `geomview` directly from the shell. These programs are distributed in the `bin/<CPU>` subdirectory of the Geomview directory, and may have been installed so that they are on your `path`.

```
In[1]:= <<OOGL.m

In[2]:= Plot3D[ Sin[x + Sin[y]], {x,-2,2}, {y,-2,2} ]

Out[2]= -SurfaceGraphics-

In[3]:= WriteChunk["mychunk",%2]
```

This writes the file `mychunk` which contains a description of the graphics object. You can then transfer this file to a system with Geomview and type

```
math2oogl < mychunk > mma.oogl
```

to convert it to the OOGL file `mma.oogl` which you can then view using Geomview. This is the equivalent of the `WriteOOGL` command.

For a result equivalent to the `Geomview` or `Show` commands, type

```
math2oogl -togeomview Mathematica geomview < mychunk
```

The `WriteRIb` command can be emulated from the shell as

```
math2oogl < mychunk | oogl2rib -n mma.tiff
```

9.5 Details of the Mathematica->Geomview Package

The `OOGL.m` package uses the external program `math2oogl` to convert `Graphics3D` objects to OOGL format, because a compiled external program is able to do this conversion many times faster than Mathematica.

The converter will sometimes handle colored SurfaceGraphics objects correctly that Mathematica does not handle correctly, which means that Geomview[object] sometimes works where Show[object] will give errors.

The converter supports the `Polygon`, `Line`, and `Point` graphics primitives, `RGBColor` `Graphics3D` directives, and `SurfaceGraphics` objects with or without `RGBColor` directives, and lists of any combination of these. It silently ignores all other directives.

The Mathematica to RenderMan conversion is actually a two-step process: Mathematica->OOGL (math2oogl), and OOGL->RenderMan (oogl2rib).

In the `WriteOOGL` and `WriteRIB` commands, filename can either be a string containing a filename, an `OutputStream` object, or a string starting with a `!` to send the output to a command. Object can be a `Graphics3D` object, a `SurfaceGraphics` object, or a list of these.

The packages work best with Mathematica 2.0 or better. With version 1.2, the Geomview display is always on the local host.

9.6 Installing the Mathematica Packages

> **BUG:** This section is out of date. `Geomview` follows the GNU-installation habits, data goes to `PREFIX/share/geomview/`, user executables to `PREFIX/bin/`, private executables to `PREFIX/libexec/geomview/`, libraries to `PREFIX/lib/` where `PREFIX` denotes the installation prefiex specified for the `TOPSRCDIR/configure` script. Please have a look at the files `TOPSRCDIR/INSTALL` and `TOPSRCDIR/INSTALL.Geomview` for installation instructions.
>
> The installed Mathematica data-files can be found in `PREFIX/share/geomview/Mathematica/`.

If Geomview is properly installed on your system according to the instructions in See Chapter 10 [Installation], page 146, then the Mathematica-to-Geomview packages should work as described here; there should be no need for additional installation procedures. In practice, however, it is sometimes necessary to taylor the installation of the Mathematica packages and/or of Geomview itself to suit the needs of a particular system. This section contains details about how the installation works; if the Mathematica-to-Geomview connection does not seem to work for you after following the Geomview installation procedure, consult this section to see what might need to be fixed.

In this section, the phrase *Geomview installation* refers any of the procedures in See Chapter 10 [Installation], page 146. The way the Mathematica packages work and are installed is the same regardless of whether you have one of the binary distributions or the source distribution.

1. The relevant mathematica files are `OOGL.m`, `Geomview.m`, and `BezierPlot.m`; Mathematica must be able to find these files. They are distributed in the `PREFIX/share/geomview/Mathematica` subdirectory of the binary distributions,

and in the `TOPSRCDIR/src/bin/geomutil/math2oogl` subdirectory of the source distribution. These files need to be in a directory that is on Mathematica's search path. You can look at the value of the `$Path` variable in a Mathematica session on your system to see a list of the directories on Mathematica's search path.

The Geomview installation procedure puts copies of the Mathematica packages into a directory that you specify (`MMAPACKAGEDIR`). This should ensure that Mathematica can find them. Alternately, you could arrange to append the pathname of the Mathmematica package subdirectory of the Geomview distribution to the `$Path` variable each time you run Mathematica.

2. The package `OOGL.m` needs to be able to invoke the programs `geomview`, `math2oogl`, and `oogl2rib`. The Geomview installation procedure installs these programs into a directory that you specify for executables (`BINDIR`). Ideally, this directory should be on your shell's `$PATH`. More specifically, it should be on the `$PATH` of the shell in which Mathematica runs; the directory `/usr/local/bin` is usually a good choice. You can see the list of directories on this path by giving the command `!echo $path` in Mathematica.

   If for some reason you can't arrange for `geomview`, `math2oogl`, and `oogl2rib` to be in a directory on the shell's `$path`, you can modify `OOGL.m` to cause it to look for them using absolute pathnames. To do this, change the definitions of the variables `$GeomviewPath` and `$GeomRoot`, which are defined near the top of the file. Change `$GeomviewPath` to the absolute pathname of the `geomview` shell script on your system. Change `$GeomRoot` to the absolute pathname of the `$GEOMROOT` directory on your system. If you do this, you should also make sure there are copies of `geomview`, `math2oogl`, and `oogl2rib` in the `$GEOMROOT/bin/<CPU>`.

3. The `geomview` shell script, which `OOGL.m` uses to invoke Geomview, needs to be able to find the geomview executable file (called `gvx`). The Geomview installation procedure should have been taken care of this, but if your Mathematica session doesn't seem to be able to invoke Geomview, it's worth double-checking that the settings in the `geomview` script are correct.

10 Installation

> **BUG:** This section is out of date. `Geomview` follows the GNU-installation habits, data goes to `PREFIX/share/geomview/`, user executables to `PREFIX/bin/`, private executables to `PREFIX/libexec/geomview/`, libraries to `PREFIX/lib/` where `PREFIX` denotes the installation prefiex specified for the `TOPSRCDIR/configure` script. Please have a look at the files `TOPSRCDIR/INSTALL` and `TOPSRCDIR/INSTALL.Geomview` for installation instructions.

What you do to install Geomview depends on which kind of computer you have and on whether you have the source distribution or the binary distribution.

In general, if you don't care about looking at Geomview's source code, you should get one of the binary distribution. The binary installation is much easier and quicker than compiling and installing the source code.

10.1 Installing the Unix Binary Distribution

> **BUG:** This section is out of date. `Geomview` follows the GNU-installation habits, data goes to `PREFIX/share/geomview/`, user executables to `PREFIX/bin/`, private executables to `PREFIX/libexec/geomview/`, libraries to `PREFIX/lib/` where `PREFIX` denotes the installation prefiex specified for the `TOPSRCDIR/configure` script. Please have a look at the files `TOPSRCDIR/INSTALL` and `TOPSRCDIR/INSTALL.Geomview` for installation instructions.

If you have just obtained a copy of the binary distribution for a Unix system (Linux, SGI, Solaris, HP, etc), you should be able to run Geomview and make use of most of its features immediately after unpacking it by `cd`'ing to the directory that it is in and typing `geomview`.

In order to fully install Geomview so that you can run it from any directory and use all of its features, follow the steps in this section. In particular, you must go through this installation procedure in order to use Geomview to display Mathematica graphics.

Geomview is distributed in a directory that contains various files and subdirectories that Geomview needs at run-time, such as data files and external modules. It also contains other things distributed with Geomview, such as documentation and (in the soure-code distribution) source-code. We refer to the root directory of this tree as the `$GEOMROOT` directory. This is the directory called `Geomview` that is created when you unpack the distribution file.

To install Geomview on your system, arrange for the `$GEOMROOT` directory to be in a permanent place. Then, in a shell window, `cd` to that directory and type `install`. This runs a shell script which does the installation after asking you several questions about where you want to install the various components of Geomview.

After running the `install` script you should now be able to run Geomview from any directory on your system. (You may need to give the `rehash` command in any shells on your computer that were started up before you did the installation.)

The `install` script puts copies of the files in `$GEOMROOT/bin/<CPU>` and `$GEOMROOT/man` into the directories you specified for executables and man pages, respectively. Once you

have done the installation you can cut down one the disk space required by Geomview by removing some files from these directories, since copies have been installed elsewhere. You should first test that your installed Geomview works properly because once you remove these files from their distribution directories you will not be able to do the installation again.

In particular, the files you can remove are

`$GEOMROOT/bin/<MACHTYPE>`:
: (where `<MACHTYPE>` is the type of system you are on, e.g. `linux`, `sgi`, `hpux`, etc). Remove all files from here except `gvx`, which is the geomview executable file. DO NOT REMOVE `gvx`. It is not installed elsewhere.

`$GEOMROOT/man`:
: You can remove all the files in this directory.

10.1.1 Details of the Unix Binary Installation

> **BUG:** This section is out of date. `Geomview` follows the GNU-installation habits, data goes to `PREFIX/share/geomview/`, user executables to `PREFIX/bin/`, private executables to `PREFIX/libexec/geomview/`, libraries to `PREFIX/lib/` where `PREFIX` denotes the installation prefiex specified for the `TOPSRCDIR/configure` script. Please have a look at the files `TOPSRCDIR/INSTALL` and `TOPSRCDIR/INSTALL.Geomview` for installation instructions.

The `install` script should be self-explanatory; just run it and answer the questions. This section gives some details for system administrators and other users who may want to know more about the installation.

The installation is actually done by `make`; the `install` script queries the user for the settings of the following `make` variables and then invokes `make install`.

`GEOMROOT`: the absolute pathname of the Geomview root directory. The `geomview` shell script, which is what users invoke to run Geomview, uses this to set various environment variables that Geomview needs. It is very important that this be an *absolute* pathname — i.e. it should start with a '/'.

`BINDIR`: a directory where executable files are installed. The `geomview` shell script goes here, as well as various other auxiliary programs that can be used in conjunction with `geomview`. This should be a directory that is on users' `$path`. These auxiliary programs are distributed in the `$GEOMROOT/bin/<MACHTYPE>` directory; if you specify this directory for `BINDIR`, they are left in that directory.

`MANDIR`: a directory where Unix manual pages are installed. These are distributed in the `$GEOMROOT/man` subdirectory; if you specify this directory for `MANDIR`, they are left in that directory.

`MMAPACKAGEDIR`:
: a directory where Mathematica packages are installed. This should be a directory that Mathematica searches for packages that it loads; you can see what directories your Mathematica searches by looking at the value of the `$Path` variable in a Mathematica session. The installation process will install some packages there which allow you to use Geomview to display Mathematica graphics.

These packages are distributed in the `$GEOMROOT/mathematica` subdirectory; if you specify this directory for `MMAPACKAGEDIR`, or if you specify the empty string for `MMAPACKAGEDIR`, the packages are left in that directory. For more details about the way these Mathematica packages connect to Geomview, see Section 9.6 [Package Installation], page 144.

10.2 Compiling and Installing the Source Code Distribution

BUG: This section is out of date. `Geomview` follows the GNU-installation habits, data goes to `PREFIX/share/geomview/`, user executables to `PREFIX/bin/`, private executables to `PREFIX/libexec/geomview/`, libraries to `PREFIX/lib/` where `PREFIX` denotes the installation prefiex specified for the `TOPSRCDIR/configure` script. Please have a look at the files `TOPSRCDIR/INSTALL` and `TOPSRCDIR/INSTALL.Geomview` for installation instructions.

The main reason to get the source code distribution is to look at and/or work with the source code. If you are only concered with *using* Geomview it is better to get the binary distribution. It takes anywhere from a few minutes to an hour or more to compile the entire source distribution, depending on what kind of computer you have.

Let `$GEOMROOT` denote the full pathname of the Geomview source code directory; this is the directory called `Geomview` that is created when you unpack the distribution. This directory contains the Geomview source code as well as various other files and subdirectories that Geomview needs when it runs.

Before doing any compilation you should edit the file `$GEOMROOT/makefiles/mk.site.default`. This file defines some `make` variables which specify your local configuration. This includes the pathnames of the directories into which Geomview will be installed, and possibly some other settings as well. There are comments in the file telling you what to do. This file is included by every Makefile in the source tree, so the settings you specify here are used throughout the source.

If you will be compiling for multiple systems, you can do them all in the same directory tree. By default the Makefiles are set up to put the objects files, libraries, and executables in directories which depend on the type of computer, so the two architectures will not interfere with each other. The Makefiles use a variable called `CPU` to determine the type of machine. Before doing any compilation you must arrange for this variable to have a value. There are two ways you can do this.

1. If you will always be compiling Geomview on the same type of computer edit the file `$GEOMROOT/makefiles/Makedefs.global` to set the `CPU` variable to one of the values `linux`, `FreeBSD`, `sgi`, `hpux`, `hpux-gcc`, `solaris`, `sun4os4` (for Suns with SunOS 4, not Solaris), `rs6000`, or `alpha`. If you're using a type of system not in this list, make up a new value for CPU, and write a `mk.<CPU>` file for it patterned after the other `mk.*` in the `makefiles` subdirectory.
2. If you will be compiling on more than one type of computer you can set a shell environment variable named `CPU` to one of the values above and the Makefiles will inherit the value from the environment.

Note that many of the Makefiles refer to a variable called `MACHTYPE`; this variable tells which type of graphics system to compile Geomview for. The `mk.<CPU>` files set this variable for you; in most cases its value is `x11`, which specifies that Geomview should be compiled for X windows.

Once you have configured your source tree by editing the files as described above and setting the `CPU` variable, you can compile and install Geomview by typing `make install` in the `$GEOMROOT` directory. You can also type `make all`, or equivalently just `make`, to compile without installing, and then type `make install` later to install.

You can use these same `make` comands in any subdirectory in the tree to recompile and/or install a part of Geomview or a module.

If you want to modify the complier flags used during compilation, edit the file `$GEOMROOT/makefiles/Makedefs.global`; the `COPTS` variable specifies the flags passed to the C compiler (cc).

Getting Technical Support for Geomview

There are several ways to get support for Geomview.

1. Visit the Geomview web site, `http://www.geomview.org`. It contains the latest documentation, news about development, and FAQ (Frequently Asked Questions) list.
2. Send email to the geomview-users@lists.sourceforge.net mailing list. This is a mailing list for discussing any issues related to using Geomview. To join the list, send an empty note with 'subscribe' in the subject line to geomview-users-request@lists.sourceforge.net (`mailto:geomview-users-request@lists.sourceforge.net`), or visit the list web page at `http://lists.sourceforge.net/mailman/listinfo/geomview-users`.
3. Contract with Geometry Technologies for support. Geometry Technologies is a contract support and programming company that emerged from the Geometry Center, where Geomview was written. For more information visit the Geometry Technologies web site at `http://www.geomtech.com`.

Contributing to Geomview's Development

If you are interested in contributing to the development of Geomview, there are several things you can do:

1. **Volunteer programming work**.

   If you are a programmer and make an improvement to Geomview, contact the other geomview authors by sending a note to the `geomview-users` mailing list (see `http://lists.sourceforge.net/mailman/listinfo/geomview-users`).

2. **Contract with Geometry Technologies**.

   Geometry Technologies, Inc. is a consulting firm that provides contract technical support and custom programming services in the area of 3D graphics. This includes a wide range of services related to 3D graphics, included but not limited to applications involving Geomview. To the extent that resources allow, Geometry Technologies supports the developement of Geomview; in particular it hosts the `http://www.geomview.org` web site, and its staff make ongoing improvements to Geomview itself. If you are in a position to pay for technical support or custom programming work, contracting with Geometry Technologies indirectly supports Geomview. You can also contract with with Geometry Technologies to have particular features that you want added to Geomview, or to port Geomview to a new platform. For more information see Geometry Technologies web site at `http://www.geomtech.com`.

Thank you.

Function Index

!

`!, shell` 109, 129

*

`*` 109

+

`+` 109

-

`-` 109

/

`/` 109

<

`<` 109

=

`=` 109

>

`>` 109

?

`?` 110
`??` 110

|

`|, emodule-run` 110, 114

A

`all` 110
`all camera` 110
`all emodule defined` 110
`all emodule running` 110
`all geometry` 110
`and` 110
`ap-override` 110

B

`backcolor` 110
`background-image` 110
`bbox-color` 111
`bbox-draw` 111

C

`camera` 111
`camera-draw` 111
`camera-prop` 111
`camera-reset` 111
`car` 111
`cdr` 112
`clock` 112
`command` 112
`cons` 112
`copy` 112
`cursor-still` 112
`cursor-twitch` 112

D

`defun` 112
`delete` 113
`dice` 113
`dimension` 113
`dither` 113
`draw` 113
`dump-handles` 113

E

`echo` 113
`emodule-clear` 114
`emodule-define` 114
`emodule-defined` 114
`emodule-isrunning` 114
`emodule-path` 114
`emodule-run, |` 110, 114
`emodule-sort` 115
`emodule-start` 115
`emodule-transmit` 115
`escale` 115
`eval` 115
`event-keys` 115
`event-mode` 115
`event-pick` 116
`evert` 116
`exit` 116
`ezoom` 116

F

freeze 116

G

geometry 116
geomview-version 116

H

hdefine 116
hdelete 117
help 117
hmodel 117
hsphere-draw 117

I

if 117
inhibit-warning 118
input-translator 118
interest 118

L

lambda 118
let 119
lines-closer 119
load 119
load-path 119
look 120
look-encompass 120
look-encompass-size 120
look-recenter 120
look-toward 120

M

merge 121
merge-ap 121
merge-base-ap 121
merge-baseap 121
mod 121
morehelp 121

N

name-object 121
ND-axes 121
ND-color 122
ND-xform 122
ND-xform-get 122
ND-xform-set 122
new-alien 123
new-camera 123
new-center 123
new-geometry 123
new-reset 123
NeXT 123
normalization 123
not 124

O

or 124

P

pick 124
pick-invisible 125
pickable 125
position 125
position-at 125
position-toward 125
process-events 125
progn 126

Q

quit 126
quote 126

R

rawevent 126
rawpick 126
read 126
real-id 127
redraw 127
regtable 127
rehash-emodule-path 127
replace-geometry 127
rib-display 127
rib-snapshot 127

S

scale ... 128
scene ... 128
set-clock ... 128
set-conformal-refine ... 128
set-emodule-path ... 128
set-load-path ... 128
set-motionscale ... 129
setenv ... 129
setq ... 129
sgi ... 129
shell, ! ... 109, 129
sleep-for ... 129
sleep-until ... 130
snapshot ... 130
soft-shader ... 130
space ... 130
stereowin ... 130

T

time-interests ... 131
transform ... 131
transform-incr ... 132
transform-set ... 132
truncate ... 132

U

ui-cam-focus ... 132
ui-center ... 132
ui-center-origin ... 133
ui-emotion-program ... 133
ui-emotion-run ... 133
ui-freeze ... 133
ui-html-browser ... 133
ui-motion ... 133
ui-panel ... 134
ui-pdf-viewer ... 134
ui-target ... 135
uninterest ... 135
update ... 135
update-draw ... 135

W

while ... 135
window ... 135
winenter ... 135
write ... 135
write-comments ... 136
write-handle ... 136
write-sexpr ... 136

X

xform ... 136
xform-incr ... 137
xform-set ... 137

Z

zoom ... 137

Table of Contents

Introduction to Geomview **1**

Distribution **2**

Copying **3**

GNU LESSER PUBLIC LICENSE **4**

Preamble 4
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 5
How to Apply These Terms to Your New Programs 12

History of Geomview's Development **13**

Authors 13

Supported Platforms **14**

How to Pronounce "Geomview" **15**

1 Overview **16**

2 Tutorial **17**

3 Interaction **23**

3.1 Starting Geomview 23
3.2 Command Line Options 23
3.3 Basic Interaction: The Main Panel 25
3.4 Loading Objects Into Geomview 27
3.5 Using the Mouse to Manipulate Objects 28
3.5.1 Selecting a Point of Interest 32
3.6 Changing the Way Things Look 33
3.6.1 The Appearance Panel 33
3.6.2 The Materials Panel 37
3.6.3 The Lighting Panel 38
3.7 Cameras 39
3.8 Saving your work 42
3.9 The Commands Panel 44
3.10 Keyboard Shortcuts 44

4 OOGL File Formats 49

4.1 Conventions 49
4.1.1 Syntax Common to All OOGL File Formats 49
4.1.2 File Names 49
4.1.3 Vertices 49
4.1.4 *N*-dimensional Vertices 50
4.1.5 Surface normal directions 50
4.1.6 Transformation matrices 51
4.1.7 ND Transformation matrices 51
4.1.8 Binary format 51
4.1.9 Embedded objects and external-object references 52
4.1.10 Appearances 53
4.1.11 Texture Mapping 56
4.2 Object File Formats 59
4.2.1 QUAD: collection of quadrilaterals 59
4.2.2 MESH: rectangularly-connected mesh 59
4.2.3 BBOX: simple bounding boxes 60
4.2.4 Bezier Surfaces 61
4.2.5 OFF Files 62
4.2.6 VECT Files 64
4.2.7 SKEL Files 65
4.2.8 SPHERE Files 66
4.2.9 INST Files 67
4.2.9.1 INST Examples 69
4.2.10 LIST Files 70
4.2.11 TLIST Files 70
4.2.12 GROUP Files 71
4.2.13 DISCGRP Files 71
4.2.14 COMMENT Objects 71
4.3 Non-geometric objects 72
4.3.1 Appearance Objects 72
4.3.2 Image Objects 72
4.3.3 Transform Objects 76
4.3.4 ND-Transform Objects 78
4.3.5 cameras 79
4.3.6 window 81

5 Customization: `.geomview` files 83

6 External Modules 84

6.1 How External Modules Interface with Geomview 84
6.2 Example 1: Simple External Module 84
6.3 Example 2: Simple External Module with FORMS Control Panel 88
6.4 The XForms Library 92
6.5 Example 3: External Module with Bi-Directional Communication 92

6.6 Example 4: Simple Tcl/Tk Module Demonstrating Picking 101
6.7 Module Installation 105
6.7.1 Private Module Installation 105
6.7.2 System Module Installation 105

7 GCL: the Geomview Command Language .. 107

7.1 Conventions Used In Describing Argument Types 107
7.2 GCL Reference Guide 109
7.2.1 ! 109
7.2.2 < 109
7.2.3 = ... 109
7.2.4 > 109
7.2.5 * 109
7.2.6 / 109
7.2.7 + ... 109
7.2.8 - 109
7.2.9 ? 110
7.2.10 ?? 110
7.2.11 | 110
7.2.12 all 110
7.2.13 and 110
7.2.14 ap-override 110
7.2.15 backcolor 110
7.2.16 background-image 110
7.2.17 bbox-color 111
7.2.18 bbox-draw 111
7.2.19 camera 111
7.2.20 camera-draw 111
7.2.21 camera-prop 111
7.2.22 camera-reset 111
7.2.23 car 111
7.2.24 cdr 112
7.2.25 clock 112
7.2.26 command 112
7.2.27 cons 112
7.2.28 copy 112
7.2.29 cursor-still 112
7.2.30 cursor-twitch 112
7.2.31 defun 112
7.2.32 delete 113
7.2.33 dice 113
7.2.34 dimension 113
7.2.35 dither 113
7.2.36 draw 113
7.2.37 dump-handles 113
7.2.38 echo 113

7.2.39 emodule-clear ... 114
7.2.40 emodule-define ... 114
7.2.41 emodule-defined ... 114
7.2.42 emodule-isrunning ... 114
7.2.43 emodule-path ... 114
7.2.44 emodule-run ... 114
7.2.45 emodule-sort ... 115
7.2.46 emodule-start ... 115
7.2.47 emodule-transmit ... 115
7.2.48 escale ... 115
7.2.49 eval ... 115
7.2.50 event-keys ... 115
7.2.51 event-mode ... 115
7.2.52 event-pick ... 116
7.2.53 evert ... 116
7.2.54 exit ... 116
7.2.55 ezoom ... 116
7.2.56 freeze ... 116
7.2.57 geometry ... 116
7.2.58 geomview-version ... 116
7.2.59 hdefine ... 116
7.2.60 hdelete ... 117
7.2.61 help ... 117
7.2.62 hmodel ... 117
7.2.63 hsphere-draw ... 117
7.2.64 if ... 117
7.2.65 inhibit-warning ... 118
7.2.66 input-translator ... 118
7.2.67 interest ... 118
7.2.68 lambda ... 118
7.2.69 let ... 119
7.2.70 lines-closer ... 119
7.2.71 load ... 119
7.2.72 load-path ... 119
7.2.73 look ... 120
7.2.74 look-encompass ... 120
7.2.75 look-encompass-size ... 120
7.2.76 look-recenter ... 120
7.2.77 look-toward ... 120
7.2.78 merge ... 121
7.2.79 merge-ap ... 121
7.2.80 merge-base-ap ... 121
7.2.81 merge-baseap ... 121
7.2.82 mod ... 121
7.2.83 morehelp ... 121
7.2.84 name-object ... 121
7.2.85 ND-axes ... 121

7.2.86 ND-color . . . 122
7.2.87 ND-xform . . . 122
7.2.88 ND-xform-get . . . 122
7.2.89 ND-xform-set . . . 122
7.2.90 new-alien . . . 123
7.2.91 new-camera . . . 123
7.2.92 new-center . . . 123
7.2.93 new-geometry . . . 123
7.2.94 new-reset . . . 123
7.2.95 NeXT . . . 123
7.2.96 normalization . . . 123
7.2.97 not . . . 124
7.2.98 or . . . 124
7.2.99 pick . . . 124
7.2.100 pick-invisible . . . 125
7.2.101 pickable . . . 125
7.2.102 position . . . 125
7.2.103 position-at . . . 125
7.2.104 position-toward . . . 125
7.2.105 process-events . . . 125
7.2.106 progn . . . 126
7.2.107 quit . . . 126
7.2.108 quote . . . 126
7.2.109 rawevent . . . 126
7.2.110 rawpick . . . 126
7.2.111 read . . . 126
7.2.112 real-id . . . 127
7.2.113 redraw . . . 127
7.2.114 regtable . . . 127
7.2.115 rehash-emodule-path . . . 127
7.2.116 replace-geometry . . . 127
7.2.117 rib-display . . . 127
7.2.118 rib-snapshot . . . 127
7.2.119 scale . . . 128
7.2.120 scene . . . 128
7.2.121 set-clock . . . 128
7.2.122 set-conformal-refine . . . 128
7.2.123 set-emodule-path . . . 128
7.2.124 set-load-path . . . 128
7.2.125 set-motionscale . . . 129
7.2.126 setenv . . . 129
7.2.127 setq . . . 129
7.2.128 sgi . . . 129
7.2.129 shell . . . 129
7.2.130 sleep-for . . . 129
7.2.131 sleep-until . . . 130
7.2.132 snapshot . . . 130

7.2.133 soft-shader . . . 130
7.2.134 space . . . 130
7.2.135 stereowin . . . 130
7.2.136 time-interests . . . 131
7.2.137 transform . . . 131
7.2.138 transform-incr . . . 132
7.2.139 transform-set . . . 132
7.2.140 truncate . . . 132
7.2.141 ui-cam-focus . . . 132
7.2.142 ui-center . . . 132
7.2.143 ui-center-origin . . . 133
7.2.144 ui-emotion-program . . . 133
7.2.145 ui-emotion-run . . . 133
7.2.146 ui-freeze . . . 133
7.2.147 ui-html-browser . . . 133
7.2.148 ui-motion . . . 133
7.2.149 ui-panel . . . 134
7.2.150 ui-pdf-viewer . . . 134
7.2.151 ui-target . . . 135
7.2.152 uninterest . . . 135
7.2.153 update . . . 135
7.2.154 update-draw . . . 135
7.2.155 while . . . 135
7.2.156 window . . . 135
7.2.157 winenter . . . 135
7.2.158 write . . . 135
7.2.159 write-comments . . . 136
7.2.160 write-handle . . . 136
7.2.161 write-sexpr . . . 136
7.2.162 xform . . . 136
7.2.163 xform-incr . . . 137
7.2.164 xform-set . . . 137
7.2.165 zoom . . . 137

8 Non-Euclidean Geometry . . . 138

9 Mathematica Graphics in Geomview or RenderMan . . . 139

9.1 Using Mathematica to generate OOGL files . . . 139
9.2 Using Geomview as Mathematica's Default 3D Display . . . 140
9.3 Using Mathematica to generate RenderMan files . . . 141
9.4 Using Geomview and Mathematica on Different Computers . . . 142
9.4.1 Using a Networked Geomview Host . . . 142
9.4.2 Transporting Mathematica Files to Geomview by Hand . . . 143
9.5 Details of the Mathematica->Geomview Package . . . 144
9.6 Installing the Mathematica Packages . . . 144

10 Installation 146
10.1 Installing the Unix Binary Distribution 146
10.1.1 Details of the Unix Binary Installation 147
10.2 Compiling and Installing the Source Code Distribution 148

Getting Technical Support for Geomview 150

Contributing to Geomview's Development 151

Function Index 152

List of Figures

Figure 2.1: Initial Geomview display. 17
Figure 2.2: Looking at the world. 19
Figure 2.3: The Appearance Panel. 19
Figure 2.4: Color Chooser Panel. 20
Figure 2.5: The Files Panel. 21
Figure 2.6: Trefoil and Dodecahedron. 22
Figure 3.1: The Main Panel 25
Figure 3.2: The Files Panel. 27
Figure 3.3: The Load Panel. 28
Figure 3.4: The Tools Panel. 29
Figure 3.5: The Appearance Panel. 34
Figure 3.6: Color Chooser Panel. 35
Figure 3.7: The Materials Panel. 37
Figure 3.8: The Cameras Panel. 39
Figure 3.9: The Save Panel. 42
Figure 3.10: The Commands Panel. 44