

Manual do Geomview

Versão 1.9.3 do Geomview

para GNU/Linux/Unix

August 2007

Mark Phillips et al.

Tradutor para o Português do Brasil

Jorge Barros de Abreu

<http://usr.solar.com.br/~ficmatin>

Copyright © 1992-1998 The Geometry Center

Copyright © 1998-2006 Stuart Levy, Tamara Munzner, Mark Phillips

Copyright © 2006-2007 Claus-Justus Heine

Programa interativo de visualização tridimensional.

Introduction to Geomview

Geomview é um programa interativo para visualizar e manipular objetos geométricos, originalmente escrito pelos membros do estado maior do Geometry Center na Universidade de Minnesota (EUA), começando em 1991. O Geomview pode ser usado como um visualizador independente para objetos estáticos ou como um mecanismo de visualização para outros programas que produzem dinamicamente mudanças geométricas. Geomview roda sobre muitos tipos de computadores Unix, incluindo Linux, SGI, Sun, e HP. Geomview também executa com Cygwin. Esse manual descreve Geomview em sua versão 1.9.

Geomview é um *software* livre, disponível sob os termos da Licença Pública Geral Menor do GNU ; veja [\[Copying\]](#), [page 4](#) para detalhes.

Geomview e esse manual podem ser encontrados em <http://www.geomview.org>.

É permitido fazer cópias desse manual.

Se você tiver dúvidas ou comentários sobre o Geomview ou esse manual, considere inscrever-se na lista de correio eletrônico ‘[geomview-users](#)’, que é um fórum no qual usuários do Geomview comunicam-se para responder outras questões e para compartilhar notícias sobre o que eles estão fazendo com o Geomview. Os autores do Geomview participam dessa lista e algumas vezes enviam respostas a questionamentos existentes. Para assinar a lista, visite a página da lista no sítio <http://lists.sourceforge.net/mailman/listinfo/geomview-users>.

Distribution

Geomview é um *software* livre; isso significa que qualquer um é livre para usá-lo e livre para redistribuí-lo sob certas condições. Geomview não é de domínio público; é protegido por direitos autorais e existem restrições sobre sua distribuição, mas essas restrições são montadas de forma a permitir qualquer coisa que um bom cidadão colaborador possa querer fazer. O que não é permitido é para tentar prevenir outros de compartilhamento adicional de qualquer versão do Geomview que eles possam pegar de você. As condições precisas podem ser encontradas na Licença Geral Menor do GNU que acompanha o Geomview e também aparece acompanhando essa seção.

Uma forma de acessar uma cópia do Geomview é a partir de alguém que já possua o Geomview. Você não precisa perguntar por nossa permissão para fazer isso, ou informar qualquer coisa; apenas faça a cópia. Se você tiver acesso à internet, você pode pegar a mais recente versão do Geomview em <http://www.geomview.org>.

Você também pode receber Geomview quando você compra um computador. Fabricantes de computadores estão livres para distribuir cópias sob os mesmos termos que são aplicados a qualquer pessoa. Esses termos requerem que os fabricantes forneçam a você o código completo, incluindo qualquer mudanças que eles tenham feito, e permitam a você que redistribua o Geomview recebido deles nos termos usuais da Licença Pública Geral Menor do GNU. Em outras palavras, o programa deve ser livre para você quando você o receber, não apenas livre para o fabricante.

Copying

NOTA: Geomview é distribuído sob a LICENÇA PÚBLICA GERAL MENOR. Para os propósitos dessa licença nós pensamos em Geomview como se ele fosse uma "biblioteca", e dos módulos externos do Geomview como "programas que se comunicam com a biblioteca". Fazemos isso porque queremos especificamente permitir que programas proprietários e módulos usem o Geomview.

GNU LESSER PUBLIC LICENSE

Version 2.1, February 1999

Licença Pública Geral Menor do GNU

This is an unofficial translation of the GNU Lesser General Public License into Portuguese. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU LGPL—only the original English text of the GNU LGPL does that. However, we hope that this translation will help Portuguese speakers understand the GNU LGPL better.

Esta é uma tradução não-oficial da GNU Lesser General Public License para o Português. Ela não é publicada pela Free Software Foundation e não traz os termos de distribuição legal do software que usa a GNU LGPL – estes termos estão contidos apenas no texto da GNU LGPL original em inglês. No entanto, esperamos que esta tradução ajudará no melhor entendimento da GNU LGPL em Português.

Versão 2.1, Fevereiro de 1999

Copyright © 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA [Estados Unidos da América]

É permitido a qualquer pessoa copiar e distribuir cópias sem alterações deste documento de licença, sendo vedada, entretanto, sua modificação.

[Esta é a primeira versão da GPL Menor a ser lançada. Ela também constitui a sucessora da Licença Pública de Biblioteca do GNU, daí o número 2.1. da versão].

[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.]

Introdução

As licenças da maioria dos softwares são elaboradas para suprimir sua liberdade de compartilhá-los e modificá-los. As Licenças Públicas do GNU, ao contrário, têm o objetivo de assegurar sua liberdade para compartilhar e modificar softwares livres para garantir que o software seja livre para todos os seus usuários.

A presente Licença Pública Geral Menor se aplica a alguns pacotes de software especialmente designados - normalmente bibliotecas - da Free Software Foundation e de outros autores que decidam utilizá-la. Você pode utilizá-la também, mas recomendamos que antes, você analise cuidadosamente se esta licença, ou a Licença Pública Geral comum, é a melhor estratégia a ser adotada em cada caso específico, tendo como base as explicações abaixo.

Quando falamos de software livre, estamos nos referindo a liberdade de uso e não de gratuidade de preço. Nossas Licenças Públicas Gerais são elaboradas para garantir que

you have the freedom to distribute copies of free software (charging for this service if you so desire); that you receive source code or obtain it, if you wish; that you modify the software and use parts of it in new free programs; and that you have the freedom to do these things.

To protect your freedoms, it is necessary that we make restrictions that prohibit distributors from denying these freedoms to you or of asking that you renounce them. These restrictions translate into determined responsibilities that you will have to assume, if you wish to distribute copies of the library or modify it.

For example, if you distribute copies of the library, either gratuitously or for a fee, you will have to grant to your recipients all the freedoms that we are granting to you. You will have to guarantee that they, too, receive or can obtain the source code. If you link another code with the library, you must provide the complete object-files for the recipients, in such a way that they can link them again with the library after having made changes in the library and recompiled it. And you will have to display to them these terms, so that they know their freedoms.

We protect your freedoms through a method that involves two steps: (1) we establish freedoms over the library and (2) we offer you this license, which gives you the 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. In addition, if the library is modified by someone and passed on, the recipients must know that what they have is not the original version, in such a way that the reputation of the original author is not affected by problems that may be introduced by others.

In the end, software patents represent a constant threat to the existence of any free program. We want to ensure that a company cannot effectively restrict the users of a free program by having obtained a restrictive license from a patent holder. For this reason, we insist that any license of a patent obtained for any version of the library be consistent with the full freedom of use, specified in this license.

The majority of the GNU softwares, including some libraries, are covered by the GNU General Public License. The present GNU Lesser General Public License applies to determined libraries designated, being quite different from the GNU General Public License. We use this license for determined libraries, in order to permit the linking of these libraries with non-free programs.

When a program is linked to a library, either statically or using a shared library, this combination of the two is in legal terms a combined work, a derivative of the original library. For this reason, the GNU General Public License only permits this linking if the combination as a whole satisfies its criteria of freedom. The GNU Lesser General Public License permits more flexible criteria for the linking of other codes to the library.

We call this license the GNU Lesser General Public License "Lesser" because it does less to protect the freedom of the user than the GNU General Public License. It also offers to other developers of free software a lesser advantage in competition with non-free programs. These disadvantages are the reason why we use the GNU Lesser General Public License.

Geral comum para muitas bibliotecas. Por outro lado, em determinadas circunstâncias especiais, a licença Menor oferece vantagens.

Por exemplo, em raras ocasiões, pode existir uma necessidade especial de se incentivar a mais ampla utilização possível de uma determinada biblioteca, para que ela se torne um padrão de fato. Para conseguir isso, deve-se permitir que programas não-livres utilizem a biblioteca. Um caso mais freqüente ocorre quando uma biblioteca livre desempenha a mesma função de bibliotecas não-livres amplamente usadas. Nesse caso, existem poucas vantagens em restringir a biblioteca livre somente para software livre, então utilizamos a Licença Pública Geral Menor.

Em outros casos, a permissão para usar uma determinada biblioteca em programas não-livres possibilita que um maior número de pessoas use um amplo leque de softwares livres. Por exemplo, a permissão para usar a Biblioteca C do GNU permite que muito mais pessoas usem todo o sistema operacional do GNU, bem como sua variante, o sistema operacional do GNU/Linux.

Mesmo protegendo a liberdade dos usuários em menor grau, a Licença Pública Geral Menor garante ao usuário de um programa que esteja ligado à Biblioteca a liberdade e os meios para executar o programa, usando uma versão modificada da Biblioteca.

Seguem abaixo os termos e condições exatos para a cópia, distribuição e modificação. Preste muita atenção à diferença entre uma "obra baseada na biblioteca" e uma "obra que usa a biblioteca". O primeiro contém código que é derivado da biblioteca, enquanto o segundo tem de ser combinado à biblioteca para que possa ser executado.

TERMOS E CONDIÇÕES PARA CÓPIA, DISTRIBUIÇÃO E MODIFICAÇÃO

0. O presente Contrato de Licença se aplica a qualquer biblioteca de software ou a outro programa que contenha um aviso colocado pelo titular dos direitos autorais ou outra parte autorizada, informando que ela pode ser distribuída nos termos desta Licença Pública Geral Menor (também denominada "esta Licença"). Cada licenciado doravante será denominado "você".

Uma "biblioteca" significa uma coleção de funções de software e/ou dados preparados, de forma a serem convenientemente ligados com programas de aplicação (que usam algumas dessas funções e dados) para formar executáveis.

O termo "Biblioteca", abaixo, refere-se a qualquer biblioteca de software ou obra que tenha sido distribuída de acordo com esses termos. Uma "obra baseada na Biblioteca" significa tanto a Biblioteca como qualquer obra derivada, nos termos da legislação autoral: isto é, uma obra contendo a Biblioteca ou parte dela, seja sem alterações ou com modificações e/ou traduzida diretamente para outra linguagem. (Doravante, o termo "modificação" inclui, sem reservas, o termo "tradução").

O "código-fonte" de uma obra significa o formato preferencial da obra para que sejam feitas modificações na mesma. Para uma biblioteca, o código-fonte completo significa todo o código fonte para todos os módulos contidos na mesma, além de quaisquer arquivos de definição de interface associados, além dos scripts utilizados para controlar a compilação e a instalação da biblioteca.

Outras atividades que não a cópia, distribuição e modificação não são cobertas por esta Licença; elas estão fora de seu escopo. O ato de executar um programa usando a Biblioteca não tem restrições, e o resultado gerado a partir desse programa encontra-se coberto somente se seu conteúdo constituir uma obra baseada na Biblioteca (independente do uso da Biblioteca em uma ferramenta para escrevê-lo). Na verdade, isto dependerá daquilo que a Biblioteca faz e o que o programa que usa a biblioteca faz.

1. Você pode copiar e distribuir cópias sem alterações do código-fonte completo da Biblioteca ao recebê-lo, em qualquer meio ou mídia, desde que publique, ostensiva e adequadamente, um aviso de direitos autorais (ou copyright) apropriado e uma notificação sobre a exoneração de garantias; mantenha intactas as informações, avisos ou notificações referentes a esta Licença e à ausência de qualquer garantia; e distribua uma cópia desta Licença junto com a Biblioteca.

Você poderá cobrar um valor pelo ato físico de transferir uma cópia, e você pode oferecer, se quiser, a proteção de uma garantia em troca de um valor.

2. Você pode modificar sua cópia ou cópias da Biblioteca ou qualquer parte dela, formando, assim, uma obra baseada na Biblioteca, bem como copiar e distribuir essas modificações ou obra, em conformidade com a Cláusula 1 acima, desde que atenda, ainda, a todas as seguintes condições:

- a. A obra modificada tem de ser, por si só, uma biblioteca de software.
- b. Você tem de fazer com que os arquivos modificados contenham avisos, em destaque, de que você modificou os arquivos e a data de qualquer modificação.
- c. Você tem de fazer com que a obra como um todo seja licenciada, sem nenhum custo, a todos os terceiros, de acordo com esta Licença.
- d. Se um dispositivo, na Biblioteca modificada, se referir a uma função ou a uma tabela de dados a ser fornecida por um programa de aplicação que usa esse dispositivo, outro que não um argumento transmitido quando o dispositivo é invocado, nesse caso, você terá de fazer um esforço de boa-fé para assegurar que, no caso de uma aplicação que não forneça essa função ou tabela, o dispositivo ainda assim opere, e irá realizar qualquer parte de sua finalidade que permanecer significativa.

(Por exemplo, uma função de uma biblioteca para computar raízes quadradas tem uma finalidade que é completamente bem definida independentemente da aplicação. Por essa razão, a letra d, da Cláusula 2, exige que qualquer função ou tabela fornecida pela aplicação, usada por essa função, tem de ser opcional: se a aplicação não fornecê-la, a função de raízes quadradas deverá ainda assim computar raízes quadradas).

Essas exigências se aplicam à obra modificada como um todo. Se partes identificáveis dessa obra não forem derivadas da Biblioteca e puderem ser consideradas razoavelmente, em si, como obras independentes e separadas, nesse caso, esta Licença e seus termos não se aplicarão a essas partes quando você distribui-las como obras separadas. Todavia, quando você distribuir essas mesmas partes como partes de um todo, que por si seja uma obra baseada na Biblioteca, a distribuição desse todo deverá ser realizada de acordo com esta Licença, cujas respectivas permissões para outros licenciados estendem-se à integralidade deste todo, dessa forma, a toda e qualquer parte, independentemente de quem a escreveu.

Assim, esta cláusula não tem a intenção de afirmar direitos ou contestar os seus direitos sobre uma obra escrita inteiramente por você; a intenção é, antes, de exercer o direito de controlar a distribuição de obras derivadas ou obras coletivas baseadas na Biblioteca.

Além disto, a simples agregação de outra obra, que não seja baseada na Biblioteca, à Biblioteca (ou a uma obra baseada na Biblioteca) em um volume de meio ou mídia de armazenamento ou distribuição, não inclui esta outra obra no âmbito desta Licença.

3. Você poderá optar por aplicar os termos da Licença Pública Geral do GNU ao invés desta Licença, para uma determinada cópia da Biblioteca. Para tanto, você deverá alterar todos os avisos ou notificações que se refiram a esta Licença, para que eles se refiram à Licença Pública Geral comum do GNU, versão 2, ao invés desta Licença. (Se uma versão mais nova do que a versão 2 da Licença Pública Geral comum do GNU tiver sido gerada, então você poderá especificar essa versão, se preferir). Não faça nenhuma outra alteração nesses avisos ou notificações.

Uma vez que essa alteração tenha sido feita em uma determinada cópia, ela é irreversível para esta cópia, passando a Licença Pública Geral comum do GNU a ser aplicada para todas as cópias e obras derivadas subsequentes, feitas a partir dessa cópia.

Essa opção é útil quando você desejar copiar parte do código da Biblioteca em um programa que não seja uma biblioteca.

4. Você poderá copiar e distribuir a Biblioteca (ou uma parte ou obra derivada dela, de acordo com a Cláusula 2) em código-objeto ou formato executável, sob as Cláusulas 1 e 2 acima, desde que inclua todo o código-fonte correspondente, passível de leitura pela máquina, que deve ser distribuído sob os termos das Cláusulas 1 e 2 acima, em um meio ou mídia costumeiramente utilizado para o intercâmbio de software.

Se a distribuição do código-objeto for feita pela oferta de acesso para cópia a partir de um local designado, então a permissão de acesso equivalente para copiar o código-fonte a partir do mesmo local atende a exigência de distribuição do código-fonte, mesmo que terceiros não sejam levados a copiar a fonte junto com o código-objeto.

5. Um programa que não contenha nenhum derivativo de qualquer parte da Biblioteca, mas que seja desenhado para operar com a Biblioteca ao ser compilado ou ligado a ela, é chamado de uma "obra que usa a Biblioteca". Essa obra, isoladamente, não é uma obra derivada da Biblioteca e, portanto, fica de fora do âmbito desta Licença.

Entretanto, a ligação de uma "obra que usa a Biblioteca" com a Biblioteca constitui um executável que é um derivado da Biblioteca (pois contém partes da Biblioteca), e não uma "obra que usa a Biblioteca". O executável é, assim, coberto por esta Licença. A Cláusula 6 estabelece os termos para a distribuição desses executáveis.

Quando uma "obra que usa a Biblioteca" usar material de um arquivo de cabeçalho que é parte da Biblioteca, o código-objeto para a obra poderá ser uma obra derivada da Biblioteca, mesmo que o código-fonte não o seja. Para que isto seja verdade, é especialmente importante se a obra pode ser ligada sem a Biblioteca, ou se a obra é, em si mesma, uma biblioteca. O limiar para que isto seja verdade não é definido com precisão pela lei.

Se um arquivo-objeto usar somente parâmetros numéricos, layouts e accessors da estrutura de dados, bem como pequenas macros e pequenas funções inline (dez linhas ou menos de extensão), então o uso do arquivo-objeto não é restrito, independente de

ser ele legalmente uma obra derivada. (Executáveis contendo este código-objeto mais partes da Biblioteca continuam submetidos aos termos da Cláusula 6).

Do contrário, se a obra for um derivado da Biblioteca, você poderá distribuir o código objeto da obra sob os termos da Cláusula 6. Quaisquer executáveis contendo esta obra também se submetem à Cláusula 6, estejam ou não diretamente ligados à Biblioteca em si.

6. Como exceção à Cláusula acima, você também pode combinar ou ligar uma "obra que usa a Biblioteca" à Biblioteca para produzir uma obra contendo partes da Biblioteca e distribuí-la de acordo com os termos de sua escolha, desde que estes termos permitam modificações na obra para uso próprio por parte do cliente e engenharia reversa para depuração dessas modificações.

Em cada cópia da obra, você terá de colocar um aviso, em destaque, de que a Biblioteca foi usada e que ela e seu uso estão cobertos por esta Licença. Você deverá fornecer uma cópia desta Licença. Se, durante a execução, a obra exibir avisos ou notificações de direitos autorais (ou copyright), você terá de incluir, entre eles, o aviso de direitos autorais (ou copyright) referente à Biblioteca, bem como uma referência direcionando o usuário para a cópia desta Licença. Além disso, você deve tomar ao menos uma das seguintes providências:

- a. Incluir na obra todo o código-fonte da Biblioteca, passível de leitura pela máquina, incluindo quaisquer modificações que foram usadas na obra (as quais devem ser distribuídas conforme as Cláusulas 1 e 2 acima); e, se a obra for um executável ligado à Biblioteca, com toda a "obra que usa a Biblioteca" passível de leitura pela máquina, como código-objeto e/ou código-fonte, de modo que o usuário possa modificar a biblioteca e, depois, religar para produzir um executável modificado contendo a Biblioteca modificada. (Fica entendido que o usuário que modificar o conteúdo dos arquivos de definições da Biblioteca não necessariamente será capaz de recompilar a aplicação para usar as definições modificadas).
- b. Usar um mecanismo adequado de biblioteca compartilhada para ligar com a Biblioteca. Um mecanismo adequado é aquele que (a) usa, ao tempo da execução, uma cópia da biblioteca já presente no sistema do computador do usuário, e (2) irá operar adequadamente com uma versão modificada da biblioteca, se o usuário instalar uma, desde que a versão modificada seja compatível com a interface da versão com a qual a obra foi feita.
- c. Incluir na obra uma oferta por escrito, válida por pelo menos 3 anos, oferecendo ao mesmo usuário os materiais especificados na letra "a" da Cláusula 6 acima, por um custo não superior ao custo de fazer esta distribuição.
- d. Se a distribuição da obra for feita com a permissão de acesso para copiar, a partir de um local designado, oferecer acesso equivalente para copiar os materiais acima especificados, a partir do mesmo local.
- e. Certificar-se se o usuário já recebeu uma cópia desses materiais ou de que você já enviou uma cópia a esse usuário.

Para um executável, o formato exigido da "obra que usa a Biblioteca" deve incluir quaisquer dados e programas utilitários necessários para reprodução do executável a partir dele. Todavia, como uma exceção especial, os materiais a serem distribuídos não necessitam incluir algo que seja normalmente distribuído (tanto no formato fonte

quanto binário) com os componentes mais importantes (compilador, kernel, e assim por diante) do sistema operacional no qual executável é executado, a menos que esse componente, em si, acompanhe o executável.

Pode ocorrer que essa exigência contradiga as restrições da licença de outras bibliotecas proprietárias que normalmente não acompanham o sistema operacional. Essa contradição significa que você não pode utilizar ambas e a Biblioteca juntas em um executável distribuído por você.

7. Você pode colocar dispositivos da biblioteca que sejam uma obra baseada na Biblioteca lado-a-lado em uma única biblioteca junto com outros dispositivos de bibliotecas, desde que uma distribuição separada da obra baseada na Biblioteca e dos outros dispositivos de bibliotecas seja, de outro modo, permitida e desde que você tome uma das seguintes providências:
 - a. Incluir na biblioteca combinada uma cópia dessa obra baseada na Biblioteca sem a combinação com quaisquer outros dispositivos de biblioteca. Essa cópia tem de ser distribuída de acordo com as condições das cláusulas acima.
 - b. Junto com a biblioteca combinada, fornecer um aviso, em destaque, sobre o fato de que parte dela é uma obra baseada na Biblioteca, e explicando onde encontrar o formato não combinado incluso dessa mesma obra.
8. Você não poderá copiar, modificar, sublicenciar, ligar, ou distribuir a Biblioteca, exceto conforme expressamente disposto nesta Licença. Qualquer tentativa de, de outro modo, copiar, modificar, sublicenciar, ligar ou distribuir a Biblioteca é inválida, e automaticamente terminará seus direitos sob esta Licença. Todavia, terceiros que tiverem recebido cópias ou direitos de você, de acordo com esta Licença, não terão seus direitos rescindidos, enquanto estes terceiros mantiverem o seu pleno cumprimento.
9. Você não é obrigado a aceitar esta Licença, uma vez que você não a assinou. Entretanto, nada mais concede a você permissão para modificar ou distribuir a Biblioteca ou suas obras derivadas. Esses atos são proibidos por lei se você não aceitar esta Licença. Portanto, ao modificar ou distribuir a Biblioteca (ou qualquer obra baseada na Biblioteca), você manifesta sua aceitação desta Licença para fazê-lo, bem como de todos os seus termos e condições para cópia, distribuição ou modificação da Biblioteca ou obras nela baseadas.
10. A cada vez que você redistribuir a Biblioteca (ou qualquer obra nela baseada), o receptor automaticamente recebe uma licença do licenciante original para copiar, distribuir, ligar ou modificar a Biblioteca, sujeito a estes respectivos termos e condições. Você não poderá impor quaisquer restrições adicionais ao exercício, pelos receptores, dos direitos concedidos por este instrumento. Você não tem responsabilidade de promover o cumprimento desta licença por parte de terceiros.
11. Se, como resultado de uma sentença judicial ou alegação de violação de patente, ou por qualquer outro motivo (não restrito às questões de patentes), forem impostas a você condições (tanto através de mandado judicial, contrato ou qualquer outra forma) que contradigam as condições desta Licença, você não estará desobrigado quanto às condições desta Licença. Se você não puder atuar como distribuidor de modo a satisfazer simultaneamente suas obrigações sob esta Licença e quaisquer outras obrigações pertinentes, então, como consequência, você não poderá distribuir a Biblioteca de nenhuma forma. Por exemplo, se uma licença sob uma patente não permite a redistribuição

por parte de todos aqueles que tiverem recebido cópias, direta ou indiretamente de você, sem o pagamento de royalties, então, a única forma de cumprir tanto com esta exigência quanto com esta licença será deixar de distribuir, por completo, a Biblioteca.

Se qualquer parte desta Cláusula for considerada inválida ou não executável, sob qualquer circunstância específica, o restante da cláusula deverá continuar a ser aplicado e a cláusula, como um todo, deverá ser aplicada em outras circunstâncias.

Esta cláusula não tem a finalidade de induzir você a infringir quaisquer patentes ou direitos de propriedade, nem de contestar a validade de quaisquer reivindicações deste tipo; a única finalidade desta cláusula é proteger a integridade do sistema de distribuição do software livre, o qual é implementado mediante práticas de licenças públicas. Muitas pessoas têm feito generosas contribuições à ampla gama de software distribuído através desse sistema, confiando na aplicação consistente deste sistema; cabe ao autor/doador decidir se deseja distribuir software através de qualquer outro sistema e um licenciado não pode impor esta escolha.

Esta cláusula visa deixar absolutamente claro o que se acredita ser uma consequência do restante desta Licença.

12. Se a distribuição e/ou uso da Biblioteca for restrito em determinados países, tanto por patentes ou por interfaces protegidas por direito autoral, o titular original dos direitos autorais que colocar a Biblioteca sob esta Licença poderá acrescentar uma limitação geográfica de distribuição explícita excluindo esses países, de modo que a distribuição seja permitida somente nos países ou entre os países que não foram excluídos dessa forma. Nesse caso, esta Licença passa a incorporar a limitação como se esta tivesse sido escrita no corpo desta Licença
13. A Free Software Foundation [Fundação Software Livre] poderá de tempos em tempos publicar versões revisadas e/ou novas da Licença Pública Geral Menor. Essas novas versões serão semelhantes em espírito à presente versão, podendo, porém, ter diferenças nos detalhes, para tratar de novos problemas ou preocupações.
Cada versão recebe um número distinto de versão. Se a Biblioteca especificar um número de versão desta Licença, aplicável à Biblioteca ou a "qualquer versão posterior", você terá a opção de seguir os termos e condições tanto daquela versão como de qualquer versão posterior publicada pela Free Software Foundation. Se a Biblioteca não especificar um número de licença da versão, você poderá escolher qualquer versão já publicada pela Free Software Foundation.
14. Se você desejar incorporar partes da Biblioteca em outros programas livres cujas condições de distribuição sejam incompatíveis com estas, escreva ao autor para solicitar permissão. Para software cujos direitos autorais pertencerem à Free Software Foundation, escreva à Fundação; algumas vezes, fazemos exceções nesse sentido. Nossa decisão será guiada pelos dois objetivos de preservar a condição livre de todos os derivados de nosso software livre e de promover o compartilhamento e reutilização de softwares, de modo geral.

EXCLUSÃO DE GARANTIA

15. COMO A BIBLIOTECA É LICENCIADA SEM CUSTO, NÃO HÁ NENHUMA GARANTIA PARA A BIBLIOTECA, NO LIMITE PERMITIDO PELA LEI APLICÁVEL. EXCETO QUANDO DE OUTRA FORMA ESTABELECIDO POR

ESCRITO, OS TITULARES DOS DIREITOS AUTORAIS E/OU OUTRAS PARTES FORNECEM A BIBLIOTECA "NO ESTADO EM QUE SE ENCONTRA", SEM NENHUMA GARANTIA DE QUALQUER TIPO, TANTO EXPRESSA COMO IMPLÍCITA, INCLUINDO, DENTRE OUTRAS, AS GARANTIAS IMPLÍCITAS DE COMERCIALIZAÇÃO E ADEQUAÇÃO PARA UMA FINALIDADE ESPECÍFICA. O RISCO INTEGRAL QUANTO À QUALIDADE E DESEMPENHO DA BIBLIOTECA É ASSUMIDO POR VOCÊ. CASO A BIBLIOTECA CONTENHA DEFEITOS, VOCÊ ARCARÁ COM OS CUSTOS DE TODOS OS SERVIÇOS, REPAROS OU CORREÇÕES NECESSÁRIAS.

16. EM NENHUMA CIRCUNSTÂNCIA, A MENOS QUE EXIGIDO PELA LEI APLICÁVEL OU ACORDADO POR ESCRITO, QUALQUER TITULAR DE DIREITOS AUTORAIS OU QUALQUER OUTRA PARTE QUE POSSA MODIFICAR E/OU REDISTRIBUIR A BIBLIOTECA, CONFORME PERMITIDO ACIMA, SERÁ RESPONSÁVEL PARA COM VOCÊ POR DANOS, INCLUINDO ENTRE OUTROS QUAISQUER DANOS GERAIS, ESPECIAIS, FORTUITOS OU EMERGENTES, ADVINDOS DO USO OU IMPOSSIBILIDADE DE USO DA BIBLIOTECA (INCLUINDO, ENTRE OUTROS, PERDA DE DADOS, DADOS SENDO GERADOS DE FORMA IMPRECISA, PERDAS SOFRIDAS POR VOCÊ OU TERCEIROS OU A IMPOSSIBILIDADE DA BIBLIOTECA DE OPERAR COM QUALQUER OUTRO SOFTWARE), MESMO QUE ESSE TITULAR, OU OUTRA PARTE, TENHA SIDO AVISADO SOBRE A POSSIBILIDADE DESSES DANOS.

FINAL DOS TERMOS E CONDIÇÕES

Como Aplicar Estes Termos para Suas Novas Bibliotecas

Se você desenvolver uma nova biblioteca e quiser que ela seja da maior utilidade possível para o público, nós recomendamos fazer dela um software livre que todos possam redistribuir e modificar. Você pode fazer isto permitindo a redistribuição sob estes termos (ou, alternativamente, sob os termos da Licença Pública Geral comum)

Para fazer isto, anexe as notificações seguintes à biblioteca. É mais seguro anexá-las ao começo de cada arquivo-fonte, de modo a transmitir do modo mais eficiente a exclusão de garantia; e cada arquivo deve ter ao menos a linha de "direitos autorais reservados" e uma indicação de onde a notificação completa se encontra.

uma linha para informar o nome da biblioteca e uma breve idéia do que ela faz.
Direitos Autorais Reservados (C) <ano> nome do autor

Esta biblioteca é software livre; você pode redistribuí-la e/ou modificá-la sob os termos da Licença Pública Geral Menor do GNU conforme publicada pela Free Software Foundation; tanto a versão 2.1 da Licença, ou (a seu critério) qualquer versão posterior.

Esta biblioteca é distribuído na expectativa de que seja útil, porém, SEM NENHUMA GARANTIA; nem mesmo a garantia implícita de COMERCIALIZABILIDADE OU ADEQUAÇÃO A UMA FINALIDADE ESPECÍFICA. Consulte a Licença Pública Geral Menor do GNU para mais detalhes.

Você deve ter recebido uma cópia da Licença Pública Geral Menor do GNU junto com esta biblioteca; se não, escreva para a Free Software Foundation, Inc., no endereço 59 Temple Street, Suite 330, Boston, MA 02111-1307 USA.

Inclua também informações sobre como contatar você por correio eletrônico e por meio postal. Você também pode solicitar a seu empregador (se você for um programador) ou a sua instituição acadêmica, se for o caso, para assinar uma "renúncia de direitos autorais" sobre a biblioteca, se necessário. Segue um exemplo; altere os nomes:

A Yoyodyne Ltda., neste ato, renuncia a todos eventuais direitos autorais sobre a biblioteca 'Frob' (uma biblioteca para ajustar fechaduras), escrita por James Random Hacker.

<Assinatura de Ty Coon>, 1 de abril de 1990
Ty Coon, Presidente

Isso é tudo!

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 began 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, e Mark Phillips são os autores originais do Geomview. Celeste Fowler, Charlie Gunn, e Nathaniel Thurston também fazem contribuições significativas. Daniel Krech e Scott Wisdom fizeram o NeXTStep e a adaptação do RenderMan, e Daeron Meyer e Tim Rowley fizeram a adaptação para o X windows. Muitos outros membros do estado maior do Geometry Center, bem como muitas pessoas em muitos lugares, também contribuíram.

Mark Phillips escreveu esse manual, com ajuda substancial de Stuart Levy e Tamara Munzner. Incontáveis usuários do Geomview também foram de grande ajuda por meio da leitura do manual e indicando nossos enganos.

Supported Platforms

Geomview 1.9 pode – em princípio – compilar e executar sobre qualquer claramente recente sistemas operacionais semelhantes ao Unix. Especificamente, Geomview executa sobre Linux e sobre Cygwin (Cygwin emula um ambiente semelhante ao SystemV Unix environment sob o Microsoft Windows). Desafortunadamente Geomview compila com MacOS X (Darwin), mas aparentemente comunicações com Geomview por meio de pipes e sockets causam falha de segmentação. Sinta-se livre para consertar essa falha! Veja [\[Contributing\]](#), [page 152](#), para detalhes.

How to Pronounce “Geomview“

A palavra ‘Geomview’ é uma combinação da primeira sílaba da palavra ‘geometry’, e da palavra ‘view’. Os autores pronunciam Geomview como uma palavra oxítona, isto é, tonicidade na primeira sílaba.

GE-om-view

Algumas pessoas colocam a tonicidade na segunda sílaba, onde Geomview cai na palavra ‘geometry’, mas os autores originais, que criaram o nome, preferem a pronúncia com tonicidade na primeira sílaba.

1 Overview

O principal objetivo do Geomview é mostrar objetos cuja geometria é fornecida, permitindo controle interativo sobre detalhes tais como ponto de visão, velocidade de movimento, aparência de superfícies e linhas, e assim por diante. Geomview pode manusear qualquer número de objetos e permite controle coletivo ou separado sobre eles.

A maneira mais simples de usar Geomview é como um visualizador independente para ver e manipular objetos. Geomview pode mostrar objetos descritos em uma variedade de formatos de arquivo. Geomview é acompanhado com uma larga variedade de objetos como exemplo, e você pode criar seus próprios objetos.

Você pode também usar Geomview para manusear os dados a serem mostrados provenientes de outro programa que está sendo executado simultaneamente. Como o outro programa modifica os dados, a imagem no geomview reflete as modificações. Programas que geram objetos e utilizam o Geomview para mostrá-los são chamados *módulos externos*. Módulos externos podem controlar quase todos os aspectos do Geomview. A idéia aqui é que muitos aspectos de visualização e partes da interação de programas geométricos são independentes do conteúdo geométrico e podem ser coletados conjuntamente em uma peça simples de programa que pode ser usada em uma larga variedade de situações. O autor de um módulo externo pode concentrar-se sobre a implementação dos algoritmos desejados e deixar os aspectos de visualização ao Geomview. Geomview é acompanhado por uma coleção de módulos externos a título de exemplo, e esse manual descreve como escrever seu próprio *módulo externo*.

Geomview é o produto de um esforço no *Geometry Center* para disponibilizar um *software* de geometria interativa que é particularmente apropriado para pesquisa matemática e educação. Em particular, Geomview pode mostrar coisas no espaço hiperbólico e no espaço esférico bem como no espaço Euclidiano.

Geomview permite múltiplos objetos que são controlados independentemente e câmeras. As câmeras fornecem controle interativo para movimento, aparências (incluindo iluminação, sombreadamento, e materiais), selecionando um objeto, aresta ou nível de vértice, instantâneos em um arquivo de imagem SGI ou no formato Renderman RIB, a adição ou apagamento de objetos é possível através da manipulação direta do mouse, painéis de controle, e teclas de atalho via teclado.

Geomview suporta os seguintes tipos de dados simples: poliedros com vértices compartilhados (.off), quadriláteros, malhas retangulares, vetores, e ajustes em superfícies de Bezier de grau arbitrário incluindo ajustes racionais. Hierarquias de objetos podem ser construídas com listas de objetos e instâncias de objeto(s) transformado(s) por uma ou mais matrizes 4x4. Porções arbitrárias de modificações de hierarquias podem ser transmitidas por meio da criação de referências nomeadas.

Geomview pode mostrar saídas gráficas tridimensionais provenientes do Mathematica e do Maple.

2 Tutorial

Esse capítulo conduzirá você através de alguns usos elementares do Geomview. Trabalhando do começo ao fim desse capítulo de frente a um computador onde você pode tentar acompanhar os exemplos fornecidos aqui você pegará um pouco do que você pode fazer com Geomview.

Para iniciar o Geomview, coloque seu usuário e sua senha no computador e abra uma janela de shell. Uma janela de shell é uma janela na qual você pode digitar comandos Unix; o prompt na janela usualmente termina com um '%'. Na janela de shell (o cursor do mouse deve estar posicionado sobre a janela) digite o seguinte (Enter aqui significa pressione a tecla "Enter"):

```
geomview tetra dodec Enter
```

Esse comando inicia o geomview e chama dois objetos exemplo, um tetraedro e um dodecaedro. Após poucos segundos três janelas irão aparecer; see [Figure 2.1](#).

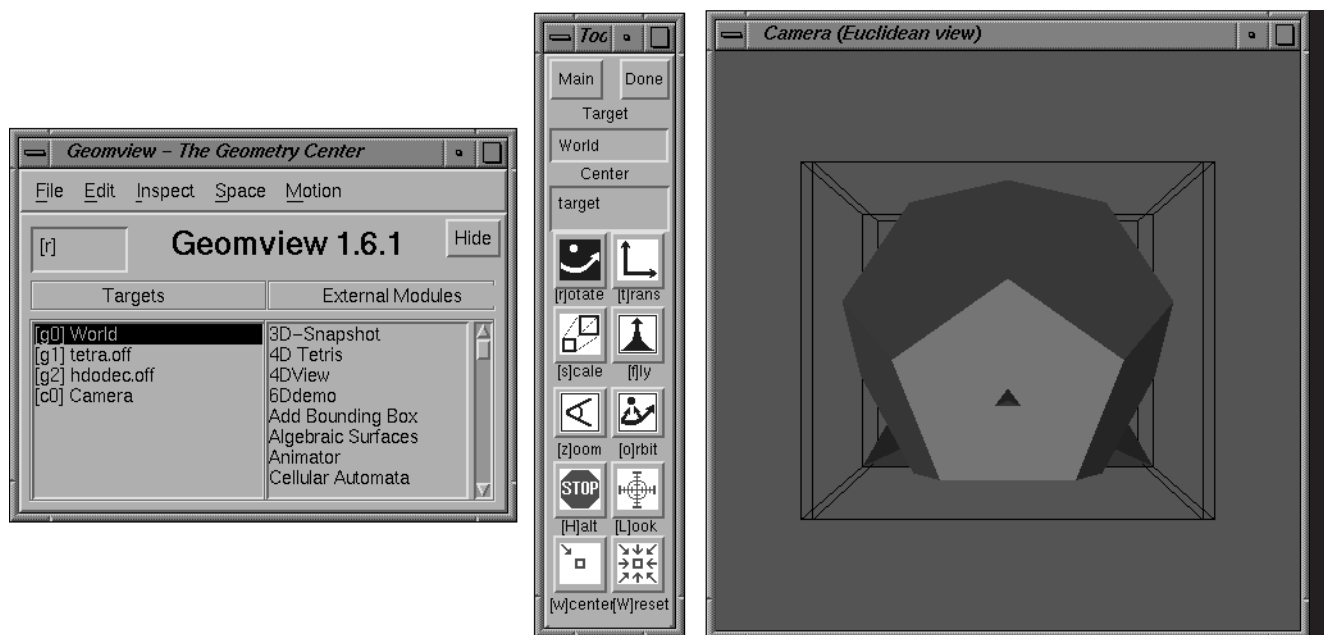


Figure 2.1: Initial Geomview display.

O painel à esquerda é o painel de controle principal do Geomview; Esse painel é chamado de painel *Main* (principal). O painel menor ao centro é o painel *Tools* (de ferramentas) e serve para selecionar diferentes tipos de movimentos. A janela do lado direito é a janela de câmera e nessa janela você vê um tetraedro largo e um dodecaedro que está parcialmente obscurecido pelo tetraedro.

Geomview tem alguns painéis mais por padrão ele mostra somente esses três. iremos descrever alguns aspectos desses três e alguns dos outros nesse tutorial. Você pode ler mais sobre esses e outros painéis nos capítulos adiante neste manual.

Coloque o cursor do mouse na janela de câmera e pressione e mantenha pressionado o botão esquerdo do mouse. Agora, enquanto mantém pressionado o botão, lentamente mova o mouse com movimentos pequenos. Você verá a figura rotacionar na direção na qual você mover o mouse. Se você liberar o botão do mouse enquanto move o mesmo, a figura continua girando. Para parar o movimento de rotação, mantenha o mouse sobre a figura e pressione rapidamente o botão esquerdo do mesmo.

Geomview utiliza o modelo da *esfera de vidro* para os movimentos iniciados através do mouse. Isso significa que você está supondo o objeto como estando dentro de uma esfera invisível e o cursor do mouse como sendo uma alça fora da esfera provida de uma ventosa. Quando você mantém pressionado o botão esquerdo do mouse, a ventosa da alça gruda na esfera; quando você libera o botão do mouse, a ventosa da alça libera a esfera. Movendo o mouse enquanto mantém pressionado o botão faz com que a esfera (e conseqüentemente o objeto) mova-se na mesma direção que o mouse.

Adicionalmente para os dois sólidos que estão atualmente na tela você pode também ver duas molduras de fios em forma de caixa na janela de câmera. Essas são as "caixas associadas" dos dois objetos. Por padrão Geomview coloca uma caixa associada em torno de cada objeto que é mostrada de forma que você tenha uma idéia de o quanto grande o objeto é.

Note que quando você move o mouse em torno do tetraedro e do dodecaedro eles se movem como se fossem uma única figura. Isso ocorre porque por padrão o que você está movendo atualmente é o "World" (mundo). Para mover um dos objetos individualmente em lugar de o mundo como um todo, mova o cursor do mouse para o navegador de alvos (*Targets*) no painel principal (*Main*). Clique (qualquer botão) sobre a palavra *tetra*. Isso faz com que o tetraedro seja o "objeto alvo". Agora mova o cursor de volta à janela de câmera e você poderá rotacionar apenas o tetraedro.

O movimento que você aplicou até agora foi a rotação, porque esse é o modo de modo de movimento selecionado no painel de ferramentas (*Tools*). Para efetuar o movimento de translação em lugar do movimento de rotação, clique sobre o botão translação (*Translate*). Agora quando você mover o mouse na janela de câmera enquanto mantém pressionado o botão esquerdo, o tetraedro (que deve ser ainda o objeto alvo de antes) irá ser transladado na direção que você move o mouse. Note que você pode transladas o tetraedro na direção da borda da janela enquanto você mantém pressionado o botão esquerdo do mouse. Se você liberar o botão do mouse enquanto move o mesmo, o tetraedro irá continuar o movimento sozinho. O tetraedro mover-se-á ao contrário do que ocorria antes muito rapidamente de forma que é muito fácil perder o rastro de onde ele se encontra.

Se você acidentalmente perder o tetraedro através de translação para muito longe da visão da janela, você pode pegá-lo de vltá através de um clique sobre o botão Centro (*Center*) no painel de ferramentas (*Tools*). Isso fará com que o tetraedro retorne para a sua posição inicial.

Clique sobre o botão Centro (*Center*) para trazer o tetraedro ao centro da janela de câmera, e então coloque-o em uma posição de forma que você possa ver completamente o dodecaedro.

Seu mundo agora tem dois objetos que estão um ao lado do outro. Você pode ver o dodecaedro no meio da janela de câmera e pode ver parte do tetraedro parcialmente fora da janela de câmera. Volte para o navegador de alvos (*Targets*) no painel principal (*Main*)

e clique sobre o "World" para selecionar o referido mundo novamente. Agora clique sobre o botão "Olhar Para" (*Look At*) no painel de ferramentas (*Tools*). Você pode ver o dodecaedro e o tetraedro ajustando-se ao meio da janela (figura see [Figure 2.2](#)). O botão "Olhar Para" (*Look At*) posiciona a câmera em uma posição tal que o objeto alvo fique centrado na janela.

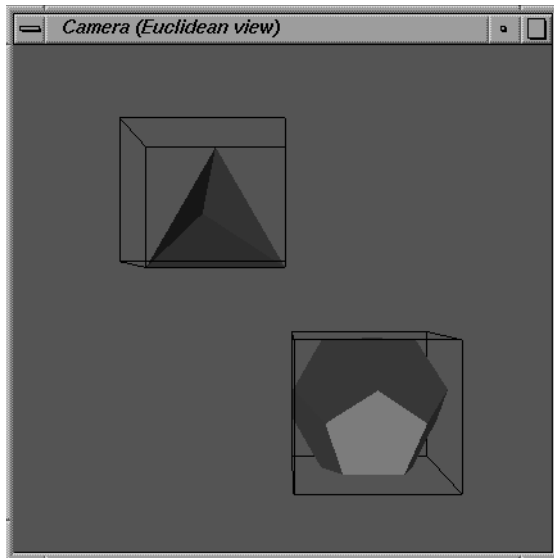


Figure 2.2: Olhando para o Mundo.

Agora coloque o cursor sobre o meio do dodecaedro e dê sobre ele um duplo clique com o botão direito do mouse. Isso significa clicar no mouse para baixo e para cima duas vezes em uma rápida sucessão. Note que o dodecaedro torna-se o objeto alvo; você pode ver isso no navegador de alvos (*Targets*) do painel principal (*Main*). Um duplo clique no botão direito do mouse sobre um objeto é outra forma de fazer esse objeto tornar-se o objeto alvo.

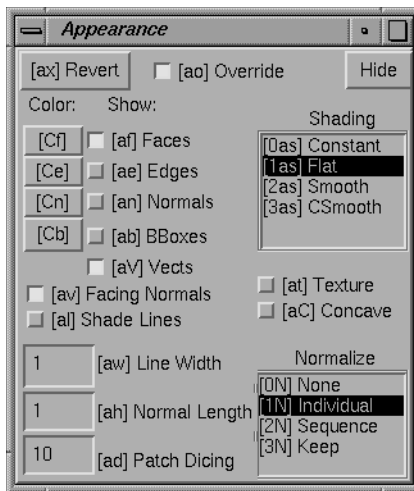


Figure 2.3: A Painel Aparência.

Vá para o menu *Inspect* no topo do painel principal (*Main*) e selecione Aparência (*Appearance*). Isso faz aparecer o painel "Aparência" (*Appearance*). Quando ele aparece, se estiver parcialmente obscurecido por outra janela do Geomview você pode movê-lo para um lado arrastando sua moldura com o botão do meio do mouse pressionado.

O painel Aparência (*Appearance*) permite a você controlar várias coisas sobre a maneira como o Geomview desenha objetos. Note os botões rotulados com *[af] Faces* e *[ae] Edges* (arestas). Clique sobre o *[ae] Edges* uma vez, e note que Geomview está agora ressaltando/destacando as arestas do dodecaedro. Clique sobre o *[ae] Edges* novamente e as arestas desaparecem. Clique muitas vezes e assista as arestas indo e voltando. Quando você tiver feito isso o suficiente, deixe as arestas habilitadas e clique sobre o botão *[af] Faces*. Essa ação alterna entre exibir ou não as faces. Clique sobre o botão novamente para de forma que a exibição das faces fique habilitada.

Agora clique sobre o botão *[Cf] Faces* sob a palavra *COLOR*. Um painel de escolha de cores aparecerá (see [Figure 2.4](#)).

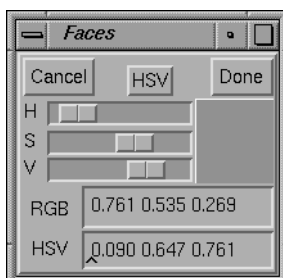


Figure 2.4: Painel de Escolha de Cores.

Note os três botões deslizantes, *H*, *S*, e *V*, controlando a matiz (*hue*), saturação, e valor (iluminação). Clicando sobre o botão *HSV* fornece um diferente conjunto de botões deslizantes, um para vermelho (*red*), outro para verde (*green*), e outro para azul (*blue*). Valores numéricos para ambos os sistemas de cores RGB e HSV podem ser vistos ou editados na parte inferior do painel. A cor inicial do dodecaedro foi especificada no arquivo ‘*dodec*’ que você chamou quando iniciamos o Geomview. A cor que você especificou com o painel de cores sobrescreveu as cores antigas. Você pode ajustar a intensidade da cor com o botão deslizante *Intensity*. Quando você encontrar uma cor que você gosta, clique sobre o botão *Done*.

Agora coloque o cursor do mouse em algum lugar sobre o fundo cinza da janela de câmera e duplo-clique no botão direito; isso seleciona "World" como objeto alvo. Clique no botão *Look At* para olhar para o mundo novamente.

Note que no painel de Aparência (*Appearance*) as escolhas dos botões se modificavam à medida que o lado esquerdo também mudava com o dodecaedro. Isso ocorre porque o painel *Appearance* sempre mostra as escolhas para o objeto alvo, que agora é o mundo, o qual ainda tem suas escolhas padrão.

Clique sobre o botão *[ab] BBox* sob a palavra *Draw*. A caixa associada desaparece. agora pona o cursor de volta na janela de câmera. No teclado, digite as teclas *a b*. Note que a caixa associada aparece novamente. *a b* é o atalho de teclado para o botão que alterna entre a exibição ou não da caixa associada; a sequência de caracteres "[ab]" aparece sobre o botão para indicar isso. A maioria dos botões do Geomview possuem atalhos de teclado que você pode usar se preferir. Isso será útil quando você estiver familiarizado com o Geomview e não quiser ter de se mover entre uma montanha de painéis.

Agora selecione o tetraedro, use qualquer das duas formas: duplo-clicando o botão direito do mouse sobre o tetraedro, ou selecionando "tetra" no navegador de alvos (*Targets*). Então clique sobre o botão *Delete* do menu *Edit* no painel principal (*Main*). O tetraedro deve desaparecer. Essa é a forma de você se livrar de um objeto.

Você pode também chamar objetos de dentro do Geomview. Clique sobre o menu *File* no painel principal (*Main*) e escolha abrir (*Open*). O painel de arquivos (*Files*) irá aparecer. Abaixo do meio desse painel, onde se lê *Path List*, temos um navegador com três linhas dentro dele; a segunda linha é um diretório com montanhas de exemplos do Geomview dentro dele. Clique sobre aquela segunda linha; see [Figure 2.5](#). Role para baixo na lista de arquivos até você ver ‘*tref.off*’. Clique sobre aquela linha, e então clique sobre o botão *OK*. Um grande tubo em forma de trevo irá aparecer em sua janela. Clique sobre o botão *Hide* no painel *Files* para dispensar o painel.

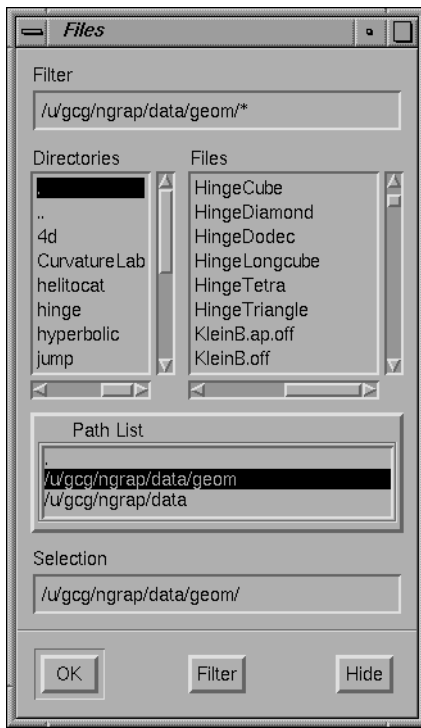


Figure 2.5: O Painel de Arquivos.

Agora clique sobre o botão *Reset* no painel de ferramentas (*Tools*). Isso fará com que todas as figuras retornem ao centro da janela de câmera. Você pode ver um dodecaedro e uma protuberância do trevo (see [Figure 2.6](#)).

Brinque com a protuberância do trevo e o dodecaedro. Faça experiências com alguns outros botões no painel de ferramentas (*Tools*). Tente colorir o trevo com o painel de aparência (*Appearance*).

Para um tutorial sobre criar seus próprios objetos para chamá-los dentro do Geomview, veja ‘doc/oogl_tour’ distribuído com Geomview. As coisas naquele arquivo irão ser incorporadas em futuras versões desse manual.

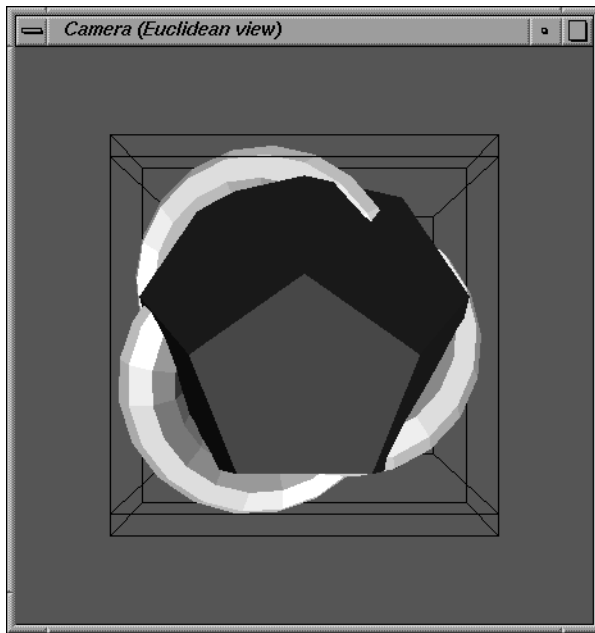


Figure 2.6: Trevo e Dodecaedro.

3 Interaction

Esse capítulo descreve como você interage com Geomview através do mouse e do teclado.

3.1 Starting Geomview

A forma usual para iniciar o Geomview é digitar *geomview* `(Enter)` em uma janela de shell (`(Enter)` significa pressionar a tecla "Enter"). Esse procedimento carrega o Geomview na memória do computador em uns poucos segundos; uma ou mais janelas irão aparecer e você pode começar a interagir com o Geomview imediatamente.

É também possível especificar ações para o Geomview executar no momento de iniciar fornecendo argumentos na linha de comando do shell. Veja [Section 3.2 \[Command Line Options\]](#), page 26.

3.2 Command Line Options

Aqui estão as opções de linha de comando que o Geomview permite:

`'-b r g b'` Escolhe a cor de fundo da janela de câmera para valores fornecidos de *r g b*.

`'-c arquivo'`

Interpreta os comandos GCL em *arquivo*, que pode ser o símbolo especial `'-'` para a entrada padrão. Paa uma descrição de GCL, veja [Chapter 7 \[GCL\]](#), page 111.

`'-c comando'`

Comandos podem também serem fornecidos literalmente, como em

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

Uma vez que *comando* inclui parêntesis, que possuem significado especial para o shell, *comando* deve receber apóstrofo. Múltiplas opções `-c` são permitidas.

`'-wins n'` Faz com que Geomview mostre inicialmente *n* janelas de câmera.

`'-wpos largura,altura[@xmin,ymax]'`

Especifica a localização inicial e o tamanho da primeira janela de câmera. Os valores para *largura*, *altura*, *xmin*, e *ymax* estão em coordenadas de tela (pixel).

`'-M[cg] [ps [un|in|in6]] PIPENOME | TCPPORT'`

A opção `'-M'` aceita modificadores: um sufixo `'g'` espera dados geométricos (o padrão), enquanto um sufixo `'c'` espera comandos GCL. Um `'p'` implica que a conexão pode usar um pipe nomeado (o padrão para tudo exceto para "NeXT"), enquanto `'s'` implica no uso de um "UNIX-domain socket" (o padrão em "NeXT"). Uma vez que na versão 1.9 do Geomview "Internet domain sockets" são também suportados; use `'sin'` para fazer o Geomview escutar uma porta IPv4 fornecida por *TCPPORT*, ou use `'sin6'` para fazer Geomview escute uma porta IPv6 (também como especificado em *TCPPORT*). `'sun'` é um sinônimo para `'s'`, i.e. use o "Unix domain socket" com o nome *PIPENOME*. Se *PIPENOME* inicia com uma barra (`'/'`), então esse nome é assumido ser um caminho absoluto, de outra forma o pipe nomeado ou socket é criado sob o diretório `'${TMPDIR}/geomview/'`.

Escutando fluxo de comando em portas TCP pode ser um risco de segurança, como Geomview por si mesmo não toma nenhum tipo de precaução de segurança, Geomview simplesmente executa todos os comandos alimentados a ele através do socket de rede. Isso também implica entrada e saída para unidades de armazenamento locais devem ser permitidas remotamente.

Exemplos:

`-M nome_de_objeto`

Mostra (possivelmente mudando dinamicamente) geometria enviada de programas `geomstuff` ou `togeomview`. Essa opção `"-M"` escuta o pipe nomeado `'/tmp/geomview/nome_de_objeto'`; você pode conseguir o mesmo efeito com os comandos de shell abaixo:

```
mkdir /tmp/geomview
```

```
mknod /tmp/geomview/nome_de_objeto p
```

(assumindo que o diretório e o pipe nomeado não existam atualmente), então executando o comando GCL:

```
(geometry nome_de_objeto < /tmp/geomview/nome_de_objeto)■
```

(see [Section 7.2.50 \[geometry\]](#), page 119)

`-Mc pipenome`

Como `'-M'` acima, mas espera comandos GCL, em lugar de dados geométricos OOGL, na conexão.

`-Mcs nome` Lê comandos a partir do "UNIX-domain socket" nomeado. `'/tmp/geomview/nome'`

`-Mcsin 40000`

Lê comandos a partir da porta IPv4 `'40000'`. Geomview por si mesmo não toma qualquer precaução de segurança, de forma que `"-Mcsin 40000"` pode ser um risco de segurança.

`'-noopengl'`

Desabilita o uso de OpenGL para (possivelmente) conversão acelerada de hardware, mesmo que o binário do Geomview tenha suporte a OpenGL compilado internamente. `"-noopengl"` também desabilita o suporte a transparência e texturas na janelas de câmera. Instantâneos `"RenderMan"` ainda terão a transparência correta e suporte a alguma textura limitada.

`'-nopanel'`

Inicia sem mostrar nenhum painel, somente a janelas gráficas. Painéis podem ser invocados mais tarde da forma usual com as teclas de atalho `Px` ou com comando `ui-panel`. Veja [Section 7.2.136 \[ui-panel\]](#), page 136.

`'-noinit'`

Não lê nenhum arquivo de inicialização. Por padrão, `geomview` lê o arquivo `'./geomview'` do systema, seguido daqueles em `'${HOME}/.geomview'` e em `'./geomview'`.

`'-e modulo'`

Inicial um módulo externo; *modulo* é o nome associado ao módulo chamado, aparecendo no painel principal no navegador de `"Applications"`, como definido pelo comando `emodule-define`. Veja [Section 7.2.34 \[emodule-define\]](#), page 117.

`'-start module args ...'`

Como `-e` mas permite a você enviar argumentos para o módulo externo. `"-` sinaliza o fim da lista de argumentos; o `"-` pode ser omitido se for o último argumento na linha de comando do Geomview.

`'-run comando-shell args ...'`

Como `"-start"` mas toma o caminho de arquivos do executável do módulo externo em lugar do nome do módulo. Os caminhos de arquivo de todos os diretórios de módulos conhecidos são anexados ao final do caminho de busca do UNIX quando for invocado o *comando-shell*.

3.3 Basic Interaction: The Main Panel

Normalmente quando você invoca Geomview, três janelas aparecem: O painel principal (*Main*), o painel de ferramentas (*Tools*), e uma janela de câmera. Geomview tem muitas outras janelas mas muitas coisas podem ser realizadas com essas três de forma que por padrão as outras não aparecem. Essa seção do manual introduz alguns conceitos básicos que são usados nas seções restantes do manual e descreve o painel principal (*Main*).

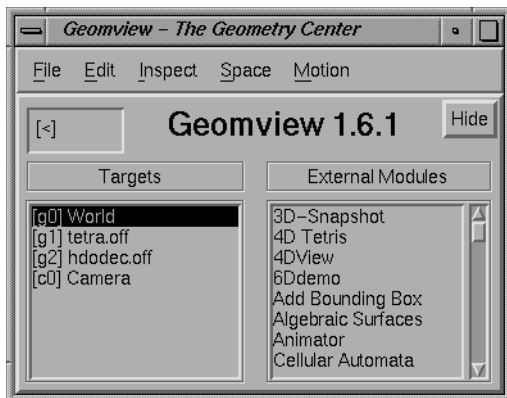


Figure 3.1: The Main Panel

Geomview pode mostrar um número arbitrário de objetos simultaneamente. O navegador *Targets* no painel principal (*Main*) mostra uma lista de todos os objetos dos quais Geomview atualmente abertos. Esse navegador tem uma linha para cada objeto que você tiver chamado, adicionalmente algumas linhas para outros objetos. Um desses outros objetos é chamado *World* e corresponde a a todos os objetos atualmente chamados, tratados como se eles fossem um objeto. A maioria das operações que você pode fazer sobre um objeto, tais como aplicar um movimento ou mudar uma cor, pode também ser feita para o objeto "World".

O navegador de alvos (*Targets*) também possui uma entrada para cada câmera. Por padrão existe somente uma câmera; é possível adicionar mais delas através da entrada *New Camera* do painel principal (*Main*) via menu *File*. Geomview trata câmeras na maioria das vezes como trata objetos geométricos. Por exemplo, você pode mover câmeras pelas proximidades e adicioná-las e apagá-las como objetos geométricos. Câmeras não são mostradas na tela como um objeto que você vê. Cada câmera tem uma janela de câmera separada que

mostra a visão como vista através da lente daquela câmera. (É passível para cada câmera mostrar uma representação geométrica de outras câmeras. Veja [Section 3.7 \[Cameras\]](#), [page 42](#).)

Devido ao fato de Geomview tratar câmeras e objetos geométricos muito similarmente, o termo *objeto* nessa documentação é usado para referir-se a qualquer dos dois indistintamente. Quando precisamos distinguir entre os dois tipos de objetos, usamos o termo "*geom*" para denotar um objeto geométrico e a palavra *câmera* para denotar uma câmera.

O objeto que está selecionado (luminosidade alta) no navegador "*Targets*" é chamado objeto alvo. Esse é o objeto que recebe quaisquer ações que você faz com o mouse ou com o teclado. Você pode mudar o objeto alvo selecionando uma linha diferente no navegador de alvos (*Targets*). Outro caminho de modificar o objeto alvo é colocar o cursor do mouse diretamente sobre um geom na janela de câmera e rapidamente dar um duplo clique no botão direito do mouse. Esse processo é chamado *selecionar*; o objeto selecionado torna-se o novo alvo.

Objetos do Geomview são todos conhecidos por dois nomes, ambos dos quais são mostrados no navegador de alvos (*Targets*). O primeiro nome lá fornecido, que aparece entre colchêtes ([]), é um nome curto atribuído pelo Geomview quando você chama o objeto. Esse nome consiste da letra 'g' para geometria e da letra 'c' para câmeras, seguindo por um número. O segundo nome é maior e mais descritivo; por padrão esse é o nome do arquivo do qual o objeto foi chamado. Os dois nomes são equivalentes até no que diz respeito ao Geomview; em qualquer ponto onde você precisar especificar um nome você pode fornecer qualquer dos dois.

Para manipular um objeto, esteja certo de que aquele objeto que você quer mover seja o objeto alvo, e coloque o cursor do mouse em uma janela de câmera. Movimentos são aplicados pressionando ou o botão esquerdo ou o botão do meio do mouse e movendo o mouse. Existem muitos modos de movimento diferentes, cada modo de movimento aplicando um diferente tipo de movimento. O navegador de modos de movimento (*MOTION MODE*) no painel principal indica o modo de movimento atual. O padrão é a rotação ("Rotate"). Você pode mudar o modo corrente de movimento selecionando um no modo de movimento no navegador de modos de movimento (*MOTION MODE*), ou usando o painel de ferramentas (*Tools*). Para maiores informações sobre modos de movimento, veja [Section 3.5 \[Mouse Motions\]](#), [page 32](#).

O navegador de módulos (*Modules*) lista módulos externos do Geomview. Um módulo externo é um programa separado que interage com Geomview para estender suas funcionalidades. Para informações sobre módulos externos, veja [Chapter 6 \[Modules\]](#), [page 88](#).

A barra de menu no topo do painel principal oferece menus para operações comuns.

Para criar novas janelas, chame novos objetos, grave os objetos ou outras informações, ou saia do geomview, veja o menu *File*.

Para copiar ou apagar objetos, veja o menu *Edit*.

Você pode chamar qualquer painel a partir do menu *Inspect*.

O menu *Space* permite a você escolher se geomview trabalha no modo Euclidiano, Hiperbolico ou Esférico. O modo Euclideano é usado por padrão. Para detalhes sobre a utilização do modo espaço *Hyperbolic* e do modo *Spherical*, veja [Chapter 8 \[Non-Euclidean Geometry\]](#), [page 140](#).

A maioria das ações que você pode fazer através dos painéis do Geomview possuem equivalentes atalhos de teclado de forma que você pode fazer a mesma ação através de digitação de uma sequência de teclas no teclado. Isso é útil para usuários avançados que estão familiarizados com as capacidades do Geomview e querem trabalhar rapidamente sem ter montanhas de painéis amontoando-se na tela. Atalhos de teclado são usualmente indicados entre colchetes ([]) próximo ao item correspondente em um apêndice. Por exemplo, o atalho de teclado para o modo *Rotate* é 'r'; isso é indicado por "[r]" que aparece antes da palavra "Rotate" no navegador *MOTION MODE*. Para usar esse atalho de teclado, apenas pressione a tecla *r* enquanto o cursor do mouse estiver em qualquer janela do Geomview. Não é necessário pressionar a tecla Enter posteriormente.

Alguns atalhos de teclado consistem em mais de uma tecla. Nesses casos apenas digite as teclas uma após a outra, sem pressionar Enter posteriormente ou entre as teclas pressionadas. Atalhos de teclado são sensíveis à caixa alta/baixa.

Muitas teclas de atalho podem ser precedidas de um parâmetro numérico. Por exemplo, digitando *ae* muda o estado do desenho de arestas, enquanto *1ae* sempre habilita o desenho de arestas.

O campo *keyboard* no canto superior esquerdo do painel principal (*Main*), imediatamente acima da palavra "Targets", ecoa o estado atual das teclas de atalho.

Para uma lista de todas as teclas de atalho, pressione a tecla ?.

3.4 Loading Objects Into Geomview

Existem muitos caminhos para chamar um objeto dentro do Geomview.

No painel de arquivos (*Files*)

Se você clicar no botão *Load* no painel principal do Geomview (*Main*), o painel de arquivos (*Files*) irá aparecer.



Figure 3.2: O Painel de Arquivos.

Esse painel permite que você selecione um arquivo a partir de uma variedade de diretórios. O topo do painel é um navegador de arquivos padrão do Motif. Abaixo deste está uma lista de diretórios no caminho de busca padrão do Geomview; clique sobre um desses para navegar entre os arquivos naquele diretório.

Para selecionar um arquivo, duplo-clique sobre o seu nome no navegador no canto superior direito, ou clique sobre o seu nome e pressione a tecla Enter, ou ainda digite o nome do arquivo dentro da caixa de texto na parte inferior do navegador e pressione a tecla Enter.

Se o arquivo selecionado contiver dados geométricos OOGL, esse arquivo irá ser adicionado ao navegador de alvos (*Targets*) do geomview. Se esse arquivo contiver comandos GCL em lugar de conter dados geométricos OOGL, o arquivo será interpretado. Veja [Chapter 7 \[GCL\], page 111](#).

Quando o apinel de arquivos (*Files*) aparecer pela primeira vez, o diretório selecionado no navegador de diretórios é o diretório atual — que corresponde ao diretório a partir do qual você chamou o Geomview. O navegador de arquivos mostra *todos* os arquivos nesse diretório, incluindo os que não são arquivos do Geomview. Se você tentar chamar u arquivo que não contenha nem um objeto OOGL nem comandos do Geomview, o Geomview irá mostrar uma mensagem de errp.

O navegador de diretórios também lista um segundo e um terceiro diretórios adicionalmente além do diretório atual. O segundo, que termina em ‘*data/geom*’, é o diretório de exemplos de dados do Geomview. Esse diretório contém uma grande variedade de amostras de objetos. Esse diretório também contém muitos subdiretórios. Em particular, os subdiretórios ‘*hyperbolic*’ e o subdiretório ‘*spherical*’ possuem amostras de objetos hiperbólicos e esféricos, respectivamente. Entradas no navegador de diretórios são vistas apenas como entradas de arquivos; para visualizar um subdiretório, clique sobre o nome do referido diretório.

O terceiro diretório mostrado no navegador de diretório, que termina em ‘*geom*’, contém muitos subdiretórios com outros arquivos do Geomview dentro deles. Esses arquivos são usados menos freqüentemente que os outros no diretório ‘*data/geom*’.

Você pode mudar a lista de diretórios mostrada no navegador de diretórios do painel de arquivos (*Files*) usando o comando `set-load-path`; Veja [Section 7.2.113 \[set-load-path\]](#), page 131.

a tecla de atalho <:

Se você digitar < em qualquer janela do Geomview, o painel *Load* irá aparecer. Esse painel é uma pequena versão do painel de arquivos (*Files*); o painel *Load* contém um campo de texto no qual você o nome de um arquivo a ser chamado (ou um comando GCL entre parêntesis). Após digitar o nome do arquivo a ser chamado, aperte a tecla `(Enter)`; Geomview irá chamar o arquivo como se você o tivesse chamado com o botão *Add* no painel de arquivos (*Files*). Se, após fazer surgir o pequeno painel *Load* com <, você decidir que quer usar o grande painel de arquivos (*Files*) após tudo, pressione o botão *File Browser*.

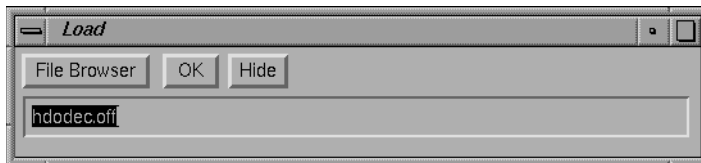


Figure 3.3: The Load Panel.

comandos para chamar objetos geométricos:

Os comandos GCL `load`, `geometry`, `new-geometry`, e `read` permitem a você chamar um objeto dentro do Geomview; veja [Chapter 7 \[GCL\]](#), page 111. Veja [Section 7.2.62 \[load\]](#), page 122. Veja [Section 7.2.83 \[new-geometry\]](#), page 125. Veja [Section 7.2.100 \[read\]](#), page 128.

3.5 Using the Mouse to Manipulate Objects

Geomview permite a você manipular objetos com o mouse. Existem seis diferentes modos de movimento do mouse: rotação (*Rotate*), translação (*Translate*), vôo da câmera (*Cam Fly*), zoom da câmera (*Cam Zoom*), homotetia de um objeto geométrico (*Geom Scale*), e órbita de câmera (*Cam Orbit*). O painel de ferramentas tem um botão para cada um desses modos;

para tracar os modos, clique sobre o botão correspondente. você pode também selecionar um modo através do navegador de modos de movimento (*Motion Mode*) no painel principal (*Main*).

Essa seção descreve a interação básica com o mouse. Para detalhes, veja [Section 3.9 \[Commands\]](#), page 48.



Figure 3.4: O Painel de Ferramentas.

Cada um dos modos de movimento usa um paradigma comum para como o movimento é aplicado. Em particular, cada modo de movimento depende do objeto alvo (*target*) atual e do atual objeto do centro (*center*). O objeto alvo atual e o atual objeto do centro são explicados nos parágrafos seguintes.

O objeto alvo atual é mostrado no campo *Target* no painel de ferramentas (*Tools*). Isso é o mesmo que o objeto selecionado no navegador de alvos (*Targets*) no painel principal (*Main*), e você pode mudar o alvo ou selecionando um novo objeto no navegador, digitando uma nova entrada no campo, ou selecionando um objeto na janela de câmera duplo-clicando no botão direito do mouse com o cursor sobre o objeto.

O atual objeto do centro é mostrado no campo *Center* no painel de ferramentas (*Tools*). Seu valor padrão é a palavra especial "target", que significa que o objeto do centro é o objeto que estiver designado como objeto alvo. Você pode mudar o objeto atual do centro para qualquer objeto digitando seu nome no campo *Center*. A origem do objeto do centro é mantido fixo no modo rotação *Rotate* e no modo *Orbit*. Normalmente o objeto do centro é um dos objetos geométricos (geoms) existentes listados no navegador de alvos (*Targets*), o centro atual das rotações é a origem daquele sistema de coordenadas daquele objeto. É

possível, todavia, selecionar um ponto arbitrário de interesse sobre um objeto como o centro. Para detalhes, veja [Section 3.5.1 \[Point of Interest\]](#), page 36.

Isso também é possível mudando o botão *BBox Center* para escolher o centro de movimento como sendo o centro do objeto atual da caixa associada. Uma vez modificado o centro da caixa geométrica ativa associada irá tornar-se o centro do movimento, se você selecionar outro objeto, então o centro do movimento irá tornar-se o centro da caixa associada à aquele objeto. Nada modificação ocorrerá quando uma câmera ou o mundo (*World*) for selecionado, você tem que digitar a palavra **target** no campo *Center* para retornar ao valor padrão.

Você aplica um movimento de mouse pressionando ou o botão esquerdo ou o botão do meio do mouse com o cursor em uma janela de câmera e movendo o mouse. A maioria dos modos de movimento possui inércia (*inertia*), que significa que se você soltar o botão enquanto move o mouse, o movimento irá continuar. Para imaginar a inércia pode ser útil imaginar o cursor do mouse como sendo uma alça; quando você pressiona um botão do mouse para baixo, o mouse agarra firmemente no objeto alvo e você pode mover esse objeto. Quando você libera o botão do mouse, a alça libera o objeto. Liberando o botão do mouse enquanto move o mesmo é como abandonar o objeto — o objeto continua movendo-se independentemente do mouse. Inércia pode ser desligada; veja o menu de movimento (*Motion*) no painel principal (*Main*), descrito abaixo.

Generally, the botão esquerdo do mouse controla movimento no plano da tela, enquanto o botão médio do mouse controla movimento ao longo ou em torno da direção de avanço.

Pressionando o tecla "shift" enquanto arrasta com o botão esquerdo ou médio do mouse na maioria dos modos de movimento fornece movimentos de baixa velocidade, útil para ajustes finos.

Você pode selecionar qualquer ponto sobre um objeto (não apenas sua origem) como centro do movimento pressionando a tecla "shift" enquanto clica no botão direito do mouse; isso escolhe o ponto de interesse.

Rotate No modo rotação (*Rotate*), pressione o botão esquerdo do mouse para rotacionar o objeto alvo em torno do objeto do centro. A rotação ocorre na direção que você move o mouse. Especificamente, o eixo de rotação passa através da origem do objeto do centro, é paralelo ao plano de visão da câmera, e é perpendicular à direção do movimento do mouse. Quando o centro for o alvo ("target"), isso significa que o objeto alvo rotaciona em torno de sua própria origem.

O botão do meio do mouse no modo de movimento tipo rotação (*Rotate*) rotaciona o objeto alvo em torno de um eixo perpendicular ao plano de visão.

Translate In *Translate* mode, hold the botão esquerdo do mouse down to translate the objeto alvo in the direction of mouse movimento. The middle mouse botão translates the target along an axis perpendicular to the view plane.

In Euclidean space, the objeto do centro 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 objeto do centro.

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 botão esquerdo do mouse down to point the câmera in a different direction. To move forward or

backward, hold down the botão do meio and move the mouse vertically. Both of these movimentos have inertia; typically the easiest way to fly around a scene is to give the câmera a slight forward push by letting go of the botão do meio while moving the mouse upward, and then using the left botão to steer.

Cam Fly affects the janela de câmera that the mouse is in; it ignores the objeto alvo and the objeto do centro.

Cam Orbit

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

In general *Cam Orbit* does not move the objeto alvo, although if the current câmera is selected as the target and the center is also the target, it will pivot that câmera 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 botão esquerdo do mouse 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 objeto alvo if that objeto is a geom. If the target is a camera, *Geom Scale* operates on the geom that was most recently the objeto alvo. Moving the mouse while holding down the botão esquerdo do mouse scales the objeto either up or down, depending on the direction of mouse movimento. The center of the applied scaling transformation is the objeto do centro.

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* botões on the *Tools* panel perform actions related to movimentos but do not change the current modo de movimento.

Stop The *Stop* botão causes all movimentos to stop. It affects all moving objetos, not just the objeto alvo. Its tecla de atalho is *H*.

The keyboard command *h*, which does not correspond to a panel botão, stops the current movimento for the objeto alvo only.

Look At The *Look At* botão causes the current câmera to be moved to a position such that it is looking at the objeto alvo, and such that the objeto alvo more or less fills the window.

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

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

Reset The *Reset* botão stops all movimento and causes all objetos to move back to their home positions.

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

The *Main* panel's *Motion* menu has special controls affecting how mouse movimentos are interpreted; the toggles are also accessible through a GCL command. See [Section 7.2.135 \[ui-motion\]](#), page 135.

[ui] *Inertia*

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

[uc] *Constrain Motion*

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

[uo] *Own Coordinates*

It's sometimes handy to move objetos 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 movimentos are interpreted that way: dragging the mouse rightward in translate mode moves the objeto in its own +X direction, and so on. May be especially useful in conjunction with the *Constrain Motion* botão.

3.5.1 Selecting a Point of Interest

It is sometimes useful to specify a particular point on some objeto in a geomview window as the center point for mouse movimentos. You can do this by shift-clicando the botão direito do mouse (i.e. clique 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 objeto.

Selecting a point of interest simplifies examining a small portion of a larger objeto. Shift-right-clique sobre o an interesting point, and select *Orbit* mode. Use the botã do meio do mouse 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 objeto do centro changes to an objeto named "CENTER", which is an invisible objeto located at the point of interest. In addition, mouse movimentos 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-clicando the botão direito do mouse again. You can cancel the point of interest altogether by shift-clicando the right botão do mouse with the cursor on the background (i.e. not on any objeto). This changes the objeto do centro back to its default value, "target".

The objeto named "CENTER", which serves as the objeto do centro 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 as the geometry for CENTER. See [Section 7.2.50 \[geometry\]](#), page 119.

What happens internally when you select a point of interest is that the center is set to the objeto called CENTER, and that objeto is positioned at the point of interest. In addition, in order for mouse movements 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 objeto's appearance, that objeto 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 objetos 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 objeto alvo.

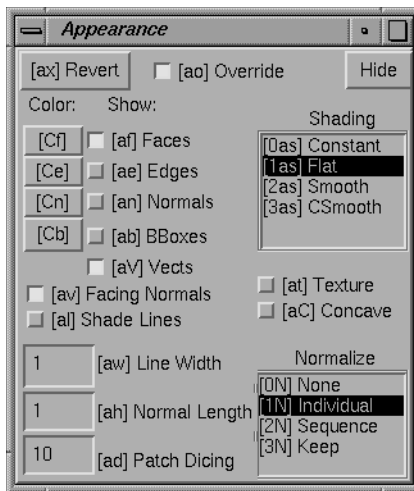


Figure 3.5: The Appearance Panel.

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 botões near the upper right corner under the word *Draw* control what parts of the target geom are drawn.

- | | |
|----------------|--|
| <i>Faces</i> | This botão specifies whether faces are drawn. |
| <i>Edges</i> | This botão specifies whether edges are drawn. |
| <i>BBox</i> | This botão specifies whether the bounding box is drawn. |
| <i>Vects</i> | This botão specifies whether VECT objetos are drawn. VECTs are a type of OOGL objeto that represent points and line segments in 3-space; they are distinct from edges of other kinds of objetos, and it is sometimes desirable to have separate control over whether they are drawn. |
| <i>Normals</i> | This botão specifies whether surface normal vectors are drawn. |

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

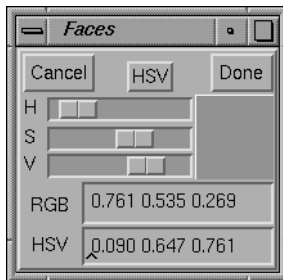


Figure 3.6: Color Chooser Panel.

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* botão 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.

Clique *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 objeto is drawn with a constant color which does not depend on the location of the face, the camera, or the light sources. If the objeto does not contain per-face or per-vertex colors, the diffuse color of the objeto's appearance is used. If the objeto contains per-face colors, they are used. If the objeto contains per-vertex colors, each face is painted using the color of its first vertex.
- Flat* Each face of the objeto 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 objeto is drawn with smoothly interpolated colors based on the normal vectors at each vertex. If the objeto does not contain per-vertex normals, this has the same effect as flat shading. If the objeto has reasonable per-vertex normals, the effect is to smooth over the edges between the faces.
- CSmooth* Each face of the objeto is drawn with exactly the specified color(s), independent of lighting, orientation, and material properties. If the objeto 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.

The *Facing Normals* botão 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 faceted objeto. Press this botão to use the normals given with the objeto.

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

3.6.2 The Materials Panel

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

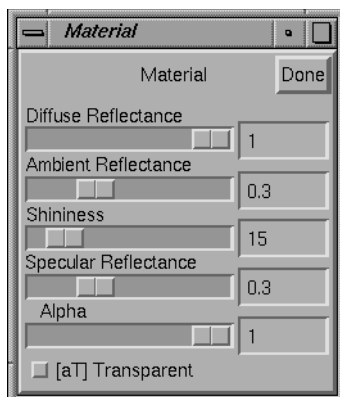


Figure 3.7: The Materials Panel.

Transparent

This botão determines whether transparency is enabled. Geomview itself does not fully support transparency yet and on some machines it does not work at all. (The missing piece is implementation of a depth-sorting algorithm in the rendering engine; not difficult, but just not done yet.) Use RenderMan if you want real transparency: when transparency is enabled, a RenderMan snapshot will contain the alpha information.

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

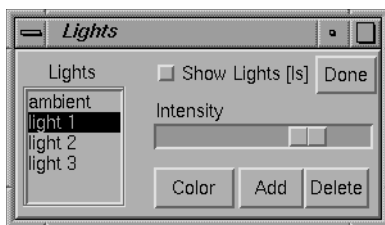
This slider controls the specular reflectance of a surface. This has to do with how directly the surface reflects light rays. Higher values give brighter specular highlights.

Done

This botão dismisses the *Materials* panel.

3.6.3 The Lighting Panel

The *Lighting* 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 objetos 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 botão brings up a color chooser which lets you select the color of the current light.

Add This botão adds a light.

Delete This botão deletes the current light.

Show Lights

This botão lets you see and change the positions of the light sources in a janela de câmera. Each light is drawn as long cylinder which is supposed to remind you of a light beam. When you clique sobre o the *Show Lights* botão Geomview goes into "light edit" mode, during which you can rotate current light by holding down the botão esquerdo do mouse and moving the mouse. Lights placed in this way are infinitely distant, so what you are changing is the angular position. Clique sobre o the *Show Lights* botão again to return to the previous modo de movimento and to quit drawing the light beams.

Done This botão 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 57. See [Section 7.2.72 \[merge-baseap\]](#), page 123. See [Section 7.2.70 \[merge-ap\]](#), page 123.

3.7 Cameras

A câmera in Geomview is the objeto that corresponds to a janela de câmera. 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 janela de câmera via the *Cameras* panel.

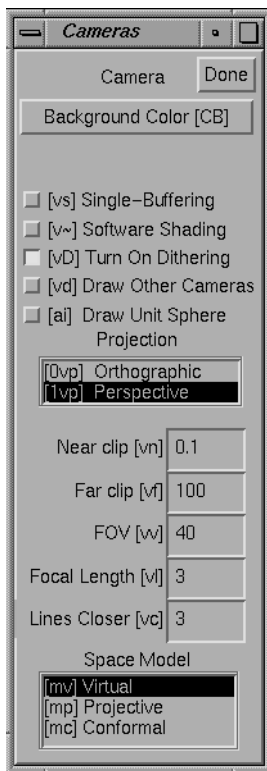


Figure 3.8: The Cameras Panel.

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

To create new janelas de câmera, use the **v+** tecla de atalho, 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 botão 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. Clique this botão to turn on correct but slower software shading.

Background Color

This botão 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 câmera 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 objetos lying at the focal distance from the camera. Focal length also affects interpretation of mouse-based translational movimentos. Speed of forward movimento (in translate, fly and orbit modes) is proportional to focal length; and objetos lying at the focal distance from the câmera translate laterally at the same rate as the mouse cursor. Finally, in N-D projection mode, câmeras 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 objeto); 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 câmera 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 140.

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 câmeras move by Euclidean movimentos. 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 câmeras move by Euclidean movimentos. 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 botão dismisses the *Cameras* panel.

3.8 Saving your work

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

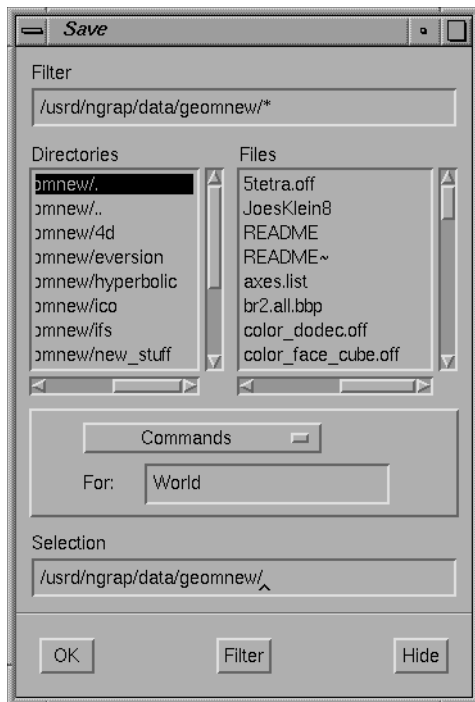


Figure 3.9: The Save Panel.

To use the *Save* panel you select the desired format in the browser next to the word *Save*, enter the name of the objeto 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 clique on the *OK* botão. When the file has been written, the *Save* panel disappears. If you want to dismiss the *Save* panel without writing a file, clique the *Cancel* botão.

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 objeto that can be written with each format is given in parentheses.

Commands (any objeto)

This write a file of GCL commands containing all information about the objeto. Loading this file later will restore the objeto 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 objeto.

Geometry [in world] (geom)

This writes an OOGL file containing just the geometry of the objeto, transformed under Geomview’s current transformation for this objeto. Use this if you have moved the objeto 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 objeto'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 '`| pnmtojpeg > snap.jpeg`' 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 objeto)

Writes an OOGL transform file giving Geomview's transform for the objeto.

Transform [to universe] (any objeto)

Writes an OOGL transform file giving a transform which is the composition of Geomview's transform for the objeto 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.

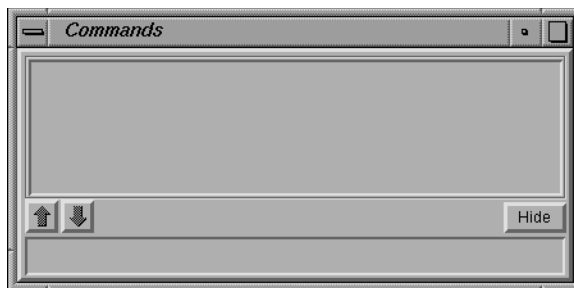


Figure 3.10: The Commands Panel.

[Move this.] Normalization is a kind of scaling; Geomview can scale an objeto so that it fits within a certain region. The main point of normalization is to allow you to easily view all of an objeto 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 objeto is to use the *Look At* botão 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.

<i>None</i>	Do no normalization.
<i>Individual</i>	Normalize this geom to fit within a unit sphere.
<i>Sequence</i>	This resembles "Individual", except when an objeto is changing. Then, "Individual" tightly fits the bounding box around the objeto whenever it changes and normalizes accordingly, while "Sequence" normalizes the union of all variants of the objeto and normalizes accordingly.
<i>Keep</i>	This leaves the current normalization transform unchanged when the objeto changes. It may be useful to apply "Individual" or "Sequence" normalization to the first version of a changing objeto to bring it in view, then switch to "Keep".

3.10 Keyboard Shortcuts

Most actions that you can do through Geomview's panels have equivalent *teclas de atalho* 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 *tecla de atalho* for *Rotate* mode is 'r'; this is indicated by "[r]" appearing before the word "Rotate" in the *MOTION MODE* browser. To use this *tecla de atalho* just hit the *r* key while the mouse cursor is in any Geomview window. You don't need to press the ⏎ or ␣ keys.

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

Keyboard commands apply while the cursor is in any *janela de câmera* and most control panels.

Many *teclas de atalho* allow numeric arguments which you type as a prefix to the command key(s). For example, the shortcut for *Near clip* in the *câmera* panel is *vn*. 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 *objeto alvo*.

<i>g</i>	world geom
<i>g#</i>	#'th geom
<i>g*</i>	All geoms
<i>c</i>	current camera
<i>c#</i>	#'th camera

c* All câmeras

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

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

The text field in the upper left corner of the *Main* panel shows the state of the current tecla de atalho.

In addition to the teclas de atalho for the panel commands, there is also a shortcut for picking a objeto alvo: type the short name of the objeto followed by *p*. For example, to select objeto *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 teclas de atalho.

Draw

<i>af</i>	Faces
<i>ae</i>	Edges
<i>an</i>	Normals
<i>ab</i>	Bounding Boxes
<i>aV</i>	Vectors

Shading

<i>0as</i>	Constant
<i>1as</i>	Flat
<i>2as</i>	Smooth
<i>3as</i>	Smooth, non-lighted
<i>aT</i>	allow transparency
<i>at</i>	texture mapping

Other

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

Color

<i>Cf</i>	faces
<i>Ce</i>	edges
<i>Cn</i>	normals
<i>Cb</i>	bounding boxes

	<i>CB</i>	background
Motions		
	<i>r</i>	rotate
	<i>t</i>	translate
	<i>z</i>	zoom FOV
	<i>f</i>	fly
	<i>o</i>	orbit
	<i>s</i>	scale
	<i>w</i>	recenter target
	<i>W</i>	recenter all
	<i>h</i>	halt
	<i>H</i>	halt all
	@	select center of movimento (e.g. <i>g 3 @</i>)
	<i>L</i>	Look At objeto
Viewing		
	<i>0vp</i>	Orthographic view
	<i>1vp</i>	Perspective view
	<i>vd</i>	Draw other views' câmeras
	<i>#vv</i>	field of View
	<i>#vn</i>	near clip distance
	<i>#vf</i>	far clip distance
	<i>v+</i>	add new camera
	<i>vx</i>	cursor on/off
	<i>vb</i>	backfacing poly cull on/off
	<i>#vl</i>	focal length
	<i>v~</i>	Software shading on/off
Panels		
	<i>Pm</i>	Main
	<i>Pa</i>	Appearance
	<i>Pl</i>	Lighting
	<i>Po</i>	Obscure
	<i>Pt</i>	Tools
	<i>Pc</i>	Câmeras

	<i>PC</i>	Commands
	<i>Pf</i>	Files
	<i>Ps</i>	Save
	<i>P-</i>	read commands from tty
	<i>PA</i>	Credits ("about")
Lights		
	<i>ls</i>	show lights
	<i>le</i>	edit lights
Space		
	<i>me</i>	Euclidean
	<i>mh</i>	Hyperbolic
	<i>ms</i>	Spherical
Model		
	<i>mv</i>	Virtual
	<i>mp</i>	Projective
	<i>mc</i>	Conformal
Other		
	<i>ON</i>	normalizaton: none
	<i>1N</i>	normalization: each
	<i>2N all</i>	normalization: all
	<i>ui</i>	movimento: Inertia
	<i>uc</i>	movimento: Constrain to axis
	<i>uo</i>	movimento: objeto's Own coordinates
	<i><</i>	
	<i>Pf</i>	load geometry/command file
	<i>dd</i>	delete objeto alvo
	<i>></i>	
	<i>Ps</i>	save state to file
	<i>TV</i>	NTSC mode toggle
	<i>p</i>	pick as objeto alvo (e.g. <i>g 3 p</i>) With no prefix, selects the objeto under the mouse cursor (like double-clicando the right mouse)

4 OOGL File Formats

The objetos that you can load into Geomview are called OOGL objetos. OOGL stands for “Objeto Oriented Graphics Library”; it is the library upon which Geomview is built.

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

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 objeto 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 objetos; See [Section 4.1.8 \[Binary format\]](#), [page 55](#), and the individual objeto descriptions.

Typical OOGL objetos begin with a key word designating objeto 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. Objeto 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 objetos 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 objeto being read from a pipe, or embedded in another OOGL objeto), all known types of objetos are tried in turn until one accepts the data as valid.

4.1.3 Vertices

Several objetos share a common style of representing vertices with optional per-vertex surface-normal and color. All vertices within an objeto 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 objeto’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’ objeto’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 objetos share a common style of representing vertices with optional per-vertex surface-normal and color. All vertices within an objeto 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 objeto’s header line)

Note, however, that N -dimensional objetos internally always have $(N+1)$ -dimensional points; the first component $x[0]$ – if present in the objeto file – is used as homogeneous divisor. This is different from the ordinary 3D case where the 4 modifier generates a 4D objeto 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 53](#), while specifying normals does not make sense.

4.1.5 Surface normal directions

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

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

When normals are not supplied with the objeto, 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 objetos incorporate 4x4 real matrices for homogeneous objeto 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 objeto, 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 objetos incorporate $(N+1) \times (N+1)$ real matrices for homogeneous objeto 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 objeto, and A $(N+1) \times (N+1)$ 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 55](#)) 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 objetos 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 63](#), See [Section 4.2.5 \[OFF\]](#), [page 66](#), and See [Section 4.2.14 \[COMMENT\]](#), [page 75](#).

Binary OOGL objetos may be freely mixed in ASCII objeto 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 objeto, though comments may appear in the header before the beginning of binary data.

4.1.9 Embedded objects and external-object references

Some objeto types (`LIST`, `INST`) allow references to other OOGL objetos, which may appear literally in the data stream, be loaded from named disk files, or be communicated from elsewhere via named objetos. 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 objeto is in a larger context, e.g. when it is part of a larger objeto 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 objeto. 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 objeto. 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 objeto. If "`NAME`" already references an objeto, then the old objeto is discarded and replaced by the new definition. See [Section 7.2.100 \[read\]](#), page 128. See [Section 7.2.52 \[hdefine\]](#), page 119.

The "<" construct causes a disk file to be read. Note that this isn't a general textual "include" mechanism; a complete OOGL objeto 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 7.2.63 \[load-path\]](#), page 122) 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 objetos can have associated "appearance" information, specifying shading, lighting, color, wire-frame vs. shaded-surface display, and so on. Appearances are inherited through objeto 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 objeto 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 objetos in their own right and can be given symbolic names and referenced by them (see [Section 4.1.9 \[References\]](#), page 56). See [Section 4.3.1 \[appearance\]](#), page 76.

Texture Mapping

There is a separate section concerning the definition of textures (see [Section 4.1.11 \[Texture Mapping\]](#), page 60).

Transparency

Rendering translucent objetos is not supported by all drawing back ends. The OpenGL renderer has limited support for it: top-level objetos (i.e. those which appear in the objeto browser of the main panel (see [Section 3.3 \[Basic Interaction\]](#), page 28) 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          # (Disable) transparency. Enabling transparency
                       # does not (necessarily) result in a correct Geomview
                       # pictures, but alpha values are used in RenderMan
```

```

# snapshots.
-normal          # (Do) draw surface-normal vectors
normscale 1      # ... with length 1.0 in objeto 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
    # objetos' colors are affected. But, objetos with the
    # "+keepcolor" attribute are immune to N-D coloring.

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

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 objetos 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 'decals'
    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 objetos contain their own appearance settings. However, if a user-supplied objeto 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 objetos, 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 57](#)). Briefly, one provides a texture image (see [Section 4.3.2 \[image\]](#), [page 76](#)), which is considered to lie in a square in (s, t) parameter space in the range $0 \leq s \leq 1$, $0 \leq t \leq 1$. 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 objeto.

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 76) }
    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 76, for the actual definition of image objetos. 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 objeto. See Section 4.3.2
\[image\], page 76.
```

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 objetos provide a more flexible way to specify texture data. See [Section 4.3.2 \[image\]](#), page 76.

`apply modulate -or- blend -or- decal`

Indicates how the texture image is applied to the surface. Here the "surface color" means the color that 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 objeto ends at end-of-file, or with a close-brace if incorporated into an objeto reference (see above).

A QUAD BINARY file format is accepted; See [Section 4.1.8 \[Binary format\]](#), page 55. The first word of binary data must be a 32-bit integer giving the number of quads in the objeto; 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 **uCMESSH**.

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 55. 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-objeto: 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 modifier 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.

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

N_d 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 $(N_u+1)*(N_v+1)$ 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), ($N_u,0$), (0, N_v), and (N_u,N_v) 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 objeto 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 55](#). 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 objetos 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 55. 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
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* objetos, each vertex has *NDim* components. For *4nSKEL* objetos, 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 55. 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 objetos
[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 140.
- ‘S’ The sphere lives in spherical space. See [Chapter 8 \[Non-Euclidean Geometry\]](#), page 140.

Sphere objetos 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` objetos; 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 $(N+1) \times (N+1)$ in the context of ND-viewing) transformation to another OOGL objeto. It begins with INST followed by these sections which may appear in any order:

`geom oogl-object`

specifies the OOGL objeto to be instantiated. See [Section 4.1.9 \[References\]](#), page 56, 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" objeto, 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 objeto (list of transformations) objeto, or a LIST whose members are ultimately TLIST objetos. In effect, the `transforms` keyword takes a collection of 4x4 matrices and replicates the *geom* objeto, 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 objeto, though a single-membered LIST would do this more efficiently.

See [Section 4.1.6 \[Transformation matrices\]](#), page 55, for the matrix format.

When replicating a single geometry by means of a TLIST objeto (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 objeto (see [Section 4.2.8 \[SPHERE\]](#), page 70) 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 $(N+1) \times (N+1)$ numbers, or there may be a reference to an `‘ntransform’` objeto, i.e.

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

or

":" symbol-representing-ntransform-object

See [Section 4.1.7 \[ND Transformation matrices\]](#), page 55, 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 objeto carries an `ntransform`. Also, if ND-viewing is active (ND-axes command, see [Chapter 7 \[GCL\]](#), page 111) then INST objetos 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 objeto; the `location` field allows putting an objeto elsewhere.

- `location global` attaches the objeto to the global (a.k.a. universe) coordinate system – the same as that in which geomview's World objetos, alien geometry, and câmeras are placed.
- `location camera` places the objeto 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 câmera space are the same as global coordinates. When a câmera is reset, the global origin is at (0,0,-3.0).
- `location ndc` places the objeto 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  -0.9 -0.9 -0.999 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 -0.999 as the Z component of the position, but that may put the objeto just nearer than the near clipping plane and make it (partially) invisible, due to floating-point error.

- `location screen` places the objeto 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 objeto 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 objetos

Syntax:

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

Note that there's no explicit separation between the oogl-objetos, 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 objeto. 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 objeto.

Syntax:

```
TLIST # key word

<4x4 matrix (16 floats)>
... # Any number of 4x4 matrices
```

TLISTs are used only within the `transforms` clause of an INST objeto. They cause the INSTs `geom` objeto to be instantiated once under each of the transforms in the TLIST. The effect is like that of a LIST of INSTs each with a single transform, and all referring to the same objeto, but is more efficient.

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

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

`myT` must be a transform objeto, 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 objeto, 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`) objeto, and replicating the objeto 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` objeto is a mechanism for encoding arbitrary data within an OOGL objeto. 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` objetos are intended to be associated with other objetos through inclusion in a `LIST` objeto. (See [Section 4.2.10 \[LIST\]](#), page 73.) The `"#"` OOGL comment syntax does not suffice for data exchange since these comments are stripped when an OOGL objeto is read in to Geomview. The `COMMENT` objeto 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 55. 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 objetos is given in the form used in See [Section 4.1.9 \[References\]](#), [page 56](#), 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 objetos of their own right, which simply means that it is possible to give them symbolic names (see [Section 4.1.9 \[References\]](#), [page 56](#)). There are other sections dealing with appearance details. See [Section 4.1.10 \[Appearances\]](#), [page 57](#).

4.3.2 Image Objects

Image objetos are used to specify pixmap data for either textures (see [Section 4.1.11 \[Texture Mapping\]](#), [page 60](#)), or for background images of câmeras (see [Section 7.2.15 \[camera\]](#), [page 115](#)).

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, <code>maxval</code> : 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, <code>maxval</code> : 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 objeto 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 objetos is like follows:

```

<image> ::=
  [ "{" ]           (curly brace, generally needed to make
                    the end of the objeto 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 enumerated 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-detect 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 filename suffix or the filter specification or by auto-detection 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

<code>'z'</code>	
<code>'gz'</code>	
<code>'gzip'</code>	data is piped through <code>'gzip -dc'</code>
<code>'bz2'</code>	
<code>'bzip2'</code>	data is piped through <code>'bzip2 -dc'</code>

Image Formats

<code>'tiff'</code>	
<code>'tif'</code>	TIFF image file format. Only supported if the <code>tifftopnm</code> executable can be found in the current execution path.
<code>'png'</code>	PNG image file format. Only supported if the <code>pngtopnm</code> executable can be found in the current execution path.
<code>'jpg'</code>	
<code>'jpeg'</code>	JPEG image file format. Only supported if the <code>jpegtopnm</code> executable can be found in the current execution path.
<code>'gif'</code>	GIF image file format. Only supported if the <code>giftopnm</code> executable can be found in the current execution path.
<code>'raw'</code>	Raw image data; the number of channels must match the number of bits set in <code>'DESTMASK'</code> . Pixels are specified with 1 byte per channel. The pixels are organized in rows as <code>'liminance[-alpha]'</code> or <code>'RGB[A]'</code> 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.bz2" 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’ specifications 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 objeto.

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

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

Syntax for a transform objeto is

```
<transform> ::=
  [ "{" ]           (curly brace, generally needed to make
                    the end of the objeto 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 objeto 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
    cantoworld { : fred }
}
```

4.3.4 ND-Transform Objects

Where – in the context of ND-viewing – a single $(N+1) \times (N+1)$ matrix is expected – as in the INST `ntransform` field, or the ND-`xform*` (see [Chapter 7 \[GCL\], page 111](#)) commands – use an `ntransform` objeto.

`ntransform` are *NRows* x *NCols* transformation matrix where usually *NRows* = *N+1* in the context of *N*-dimensional objetos and viewing. The homogeneous component of an `ntransform` sits at column zero (in contrast to ordinary `transform` objetos where it is located at column three). `ntransform` objetos operate on points of any dimension: if a point is to be transformed by an `ntransform` objeto and the dimension of the point does not match the number of rows of the `ntransform` objeto, 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` objeto is

```
<ntransform> ::=
  [ "{" ]                (curly brace, generally needed to make
                        the end of the objeto 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
        top row -- in contrast to the ordinary
        transform objetos 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 objeto -- a geometry or a camera

```

```
# cluster. Given the definition above, this puts the objeto 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 *câmera* objeto 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 objeto, typically a 4x4 matrix.

"focus" distance

Intended to suggest a typical distance from the *câmera* to the objeto of interest; used for default *câmera* positioning (the *câmera* 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 $\langle Xsize \rangle / \langle Ysize \rangle$. This normally should agree with the aspect ratio of the camera's window. Geomview normally adjusts the aspect ratio of its *câmeras* to match their associated windows.

near and far clipping plane distances

Note that both must be strictly greater than zero. Very large $\langle far \rangle / \langle near \rangle$ distance ratios cause Z-buffering to behave badly; part of an objeto 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, halfy-field is related to angular field:

$$\text{halfyfield} = \tan(\text{Y_axis_angular_field} / 2)$$

while for an orthographic one it's simply:

$$\text{halfyfield} = \text{Y_axis_linear_field} / 2$$

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" OGL objeto, 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 76](#).

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)

"bimage" { <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 objetos 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 7.2.142 \[window\]](#), page 137. See [Section 7.2.136 \[ui-panel\]](#), page 136.

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 111. See [Section 7.2.136 \[ui-panel\]](#), page 136.

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 janela de câmara 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 111. See [Section 7.2.95 \[progn\]](#), page 128. See [Section 7.2.128 \[ui-cam-focus\]](#), page 134.

6 External Modules

An external module is a program that interacts with Geomview. A module communicates with Geomview through GCL and can control any aspect 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, clique the botão esquerdo do mouse on the module's entry in the browser. While the module is running, an additional line for that module will appear in red in the browser. This line begins with a number in brackets, which indicates the *instance* 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 clicando on its red 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 109.

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; clique sobre o this entry in the browser to start the module. A surface should appear in your janela de câmera and should begin oscillating. You can stop the module by clicando on the red "[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 53. In this case, **example** is the title by which the user will see the objeto in Geomview's objeto browser, and **foo** is the internal name of the handle that the objeto 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 63.

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 111. See [Section 4.1.9 \[References\]](#), page 56. See [Section 7.2.52 \[hdefine\]](#), page 119. See [Section 7.2.100 \[read\]](#), page 128.

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 clicando on its instance line in the *Modules* browser, or else when Geomview exits.

Sometimes when you terminate this module by clicando 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" botão 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; for more information See [\[Forms\]](#), page [\[undefined\]](#).

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. Clique sobre o 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>
```

[illegible]

```

        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 objeto 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* botão; 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 botão. 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 clicado on the *Quit* botão.

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](#) 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 7.2.144 \[write\]](#), [page 137](#). This module does not do any synchronous communication.)

In asynchronous 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 shows how a module can receive user pick events, i.e. when the user cliques the botão direito do mouse with the cursor over a geom in a Geomview janela de câmera. 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 objeto 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.

Note that in order for this module to actually do anything you must have a geom loaded into Geomview and you must clique the botão direito do mouse 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 occurs. 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 OOpenGL 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 objeto 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" objeto, initially the empty list (= null objeto).
  * We replace this later upon receiving a pick event.
  */
  printf("(geometry \"pick\" { LIST } )\n");

  /* Make the "pick" objeto be non-pickable.
  */
  printf("(pickable \"pick\" no)\n");

  /* Turn off normalization, so that our pick objeto will appear in the
  * right place.
  */
  printf("(normalization \"pick\" none)\n");

  /* Don't draw the pick objeto'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 objeto. 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 External 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 endpoints 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 111](#).

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` objeto 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-objeto, that is of type `LObject *`. The last argument has the same functionality as the *block* argument, but is only invoked when picking objetos 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 7.2.54 \[help\], page 120](#)) 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 floats. A value of zero means no point was picked, i.e. the user clicado the botão direito do mouse 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` floats 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 floats: the first `pn` floats define the first vertex, and the second `pn` many floats 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 "subobjeto number 2 of subobjeto number 1 of objeto 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 received. See [Chapter 4 \[OOGL File Formats\]](#), page 53, and see [Chapter 7 \[GCL\]](#), page 111, 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 communication 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 clique).
#
# 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 objeto)
# 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 objeto 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 < hdecode.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 7.2.34 \[emodule-define\], page 117](#)). Geomview executes this command in a subshell when you clique sobre o 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 87](#). 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 111](#).

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 Geomview's `'modules/<CPU>'` directory, where `'<CPU>'` is your system type.

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, câmeras, appearances, and other basic objetos. For details of the OOGL formats, See [Chapter 4 \[OOGL File Formats\]](#), page 53. (Or equivalently, see the `oogl(5)` manual page, distributed with Geomview in the file `man/cat5/oogl.5`.)

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.

<i>cam-id</i>	is an <i>id</i> that refers to a camera.
<i>camera</i>	is an OOGL câmera specification.
<i>geom-id</i>	is an <i>id</i> that refers to a geometry.
<i>geometry</i>	is an OOGL geometry specification.
<i>id</i>	is a string which names a geometry or camera. Besides those you create, valid ones are:
<code>World, world, worldgeom, g0</code>	the collection of all geom's
<code>target</code>	selected objeto alvo (cam or geom)
<code>center</code>	selected center-of-movimento objeto
<code>targetcam</code>	last selected target camera
<code>targetgeom</code>	last selected target geom
<code>focus</code>	câmera where cursor is (or most recently was)
<code>allgeoms</code>	all geom objetos
<code>allcams</code>	all câmeras
<code>default, defaultcam, prototype</code>	future câmeras inherit default's settings

The following *ids* are used to name coordinate systems, e.g. in `pick` and `write` commands:

<code>World, world, worldgeom, g0</code>	the world, within which all other geoms live.
<code>universe</code>	the universe, in which the World, lights and câmeras live. Câmeras' world2cam transforms might better be called universe2cam, etc.
<code>self</code>	"this Geomview objeto". Transform from an objeto to <code>self</code> is the identity; writing its geometry gives the objeto itself with no enclosing transform; picked points appear in the objeto's coordinates.
<code>primitive</code>	(for <code>pick</code> 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 câmera 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 objeto is deleted. Both names are shown in the Objeto 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 7.2.117 \[shell\]](#), page 131.

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 ?

(? [command])
Gives one-line usage summary for `command`. Command may include *s as wildcards; see also [Section 7.2.73 \[morehelp\]](#), page 123. One-line command help; lists names only if multiple commands match. ? is a synonym for [Section 7.2.54 \[help\]](#), page 120

7.2.6 ??

(?? command)
`command` may include * wildcards. Prints more info than (? command). ?? is a synonym for [Section 7.2.73 \[morehelp\]](#), page 123.

7.2.7 |

| is a synonym for `emodule-run`.

7.2.8 all

`(all geometry)`
returns a list of names of all geometry objetos. Use e.g. `'(echo (all geometry))'` to print such a list.

`(all camera)`
returns a list of names of all câmeras.

`(all emodule defined)`
returns a list of all defined external modules.

`(all emodule running)`
returns a list of all running external modules.

7.2.9 and

`(and EXPR1 EXPR2)`
Evaluate `EXPR1` and `EXPR2` and return `t` if both return non-`nil`, otherwise return `nil`.

7.2.10 ap-override

`(ap-override [on|off])`
Selects whether appearance controls should override objetos' own settings. On by default. With no arguments, returns current setting.

7.2.11 backcolor

`(backcolor CAM-ID R G B)`
Set the background color of CAM-ID; R G B are numbers between 0 and 1.

7.2.12 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.

7.2.13 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.

7.2.14 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.

7.2.15 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 7.2.81 \[new-camera\]](#), page 125.

7.2.16 camera-draw

(camera-draw CAM-ID [yes|no])

Say whether or not cameras should be drawn in CAM-ID; **yes** if omitted.

7.2.17 camera-prop

(camera-prop { geometry object } [projective])

Specify the objeto to be shown when drawing other cameras. By default, this objeto is drawn with its origin at the camera, and with the camera looking toward the objeto's -Z axis. With the **projective** keyword, the camera's viewing projection is also applied to the objeto; this places the objeto'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)

7.2.18 camera-reset

(camera-reset CAM-ID)

Reset CAM-ID to its default value.

7.2.19 car

(car LIST)

returns the first element of LIST.

7.2.20 cdr

(cdr LIST)

returns the list obtained by removing the first element of LIST.

7.2.21 clock

(clock) Returns the current time, in seconds, as shown by this stream's clock. See [Section 7.2.110 \[set-clock\]](#), page 130. See [Section 7.2.119 \[sleep-until\]](#), page 131.

7.2.22 command

(command INFILE [OUTFILE])

Read commands from INFILE; send corresponding responses (e.g. anything written to filename -) to OUTFILE, stdout by default.

7.2.23 copy

(copy [ID] [name])

Copies an objeto 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.

7.2.24 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.

7.2.25 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.

7.2.26 delete

(delete ID)

Delete objeto or câmera ID.

7.2.27 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 57](#), which makes this command obsolete.

7.2.28 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.

7.2.29 dither

(dither CAM-ID {on|off|toggle})

Turn dithering on or off in that camera.

7.2.30 draw

(draw CAM-ID)

Draw the view in CAM-ID, if it needs redrawing. See [Section 7.2.102 \[redraw\]](#), [page 129](#).

7.2.31 dump-handles

(dump-handles)

Dump the list of currently active handles to stdout. This function is intended for internal debugging use only.

7.2.32 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).

7.2.33 emodule-clear

(emodule-clear)

Clears the geomview application (external module) browser.

7.2.34 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 7.2.38 \[emodule-run\]](#), page 117 for discussion of external modules.

7.2.35 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.

7.2.36 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).

7.2.37 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 7.2.112 \[set-emodule-path\]](#), page 130.

7.2.38 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 ge-

omview 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. Clicando 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 7.2.34 \[emodule-define\]](#), page 117. See [Section 7.2.40 \[emodule-start\]](#), page 118.

7.2.39 emodule-sort

```
(emodule-sort)
```

Sorts the modules in the application browser alphabetically.

7.2.40 emodule-start

```
(emodule-start NAME)
```

Starts the external module NAME, defined by emodule-define. Equivalent to clicando on the corresponding module-browser entry.

7.2.41 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 modname is not running.

7.2.42 escale

```
(escale GEOM-ID FACTOR)
```

Same as scale but multiplies by exp(scale). Obsolete.

7.2.43 event-keys

```
(event-keys {on|off})
```

Turn keyboard events on or off to enable/disable teclas de atalho.

7.2.44 event-mode

```
(event-mode MODESTRING)w
```

Set the mouse event (movimento) 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"

7.2.45 event-pick

(event-pick {on|off})

Turn picking on or off.

7.2.46 evert

(evert GEOM-ID [yes|no])

Set the normal eversion state of GEOM-ID. If the second argument is omitted, toggle the eversion state.

7.2.47 exit

(exit) Terminates geomview.

7.2.48 ezoom

(ezoom GEOM-ID FACTOR)

Same as zoom but multiplies by exp(zoom). Obsolete.

7.2.49 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.

7.2.50 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.

7.2.51 geomview-version

(geomview-version)

Returns a string representing the version of geomview that is running.

7.2.52 hdefine

(hdefine geometry|camera|window|appearance|image|transform|nttransform 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 56. See [Section 7.2.100 \[read\]](#), page 128. See [Section 7.2.53 \[hdelete\]](#), page 120.

7.2.53 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 objetos referring to the handle, but once those objetos are gone, the handle will also automatically go away. The objeto the handle refers to (if any) will only be deleted if there are no other references to that objeto.

If the optional first argument is omitted, then the first handle matching *name* will be deleted, regardless of the type of the objeto 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 function 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 56. See [Section 7.2.100 \[read\]](#), page 128. See [Section 7.2.52 \[hdefine\]](#), page 119.

7.2.54 help

```
(help [command])
```

Command may include **s* as wildcards; see also [Section 7.2.54 \[help\]](#), page 120
One-line command help; lists names only if multiple commands match.

7.2.55 hmodel

```
(hmodel CAMID {virtual|projective|conformal})
```

Set the model used to display geometry in this camera. See [Section 7.2.122 \[space\]](#), page 132.

7.2.56 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.

7.2.57 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.

7.2.58 inhibit-warning

(inhibit-warning STRING)

Inhibit warning inhibits 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.

7.2.59 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" "vrm12oogl")
```

7.2.60 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 suppresses 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 clique). See the [Section 7.2.89 \[pick\]](#), [page 126](#) command for details.

7.2.61 lines-closer

(lines-closer CAM-ID DIST)

Draw lines (including edges) closer to the camera than polygons by DIST / 10⁵ 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.

7.2.62 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 objeto; loading a camera opens a new window; loading a command file executes those commands.

7.2.63 load-path

`(load-path)`

Returns the current search path for command, geometry, etc. files. Note: to actually see the value returned by this function you should wrap it in a call to `echo: (echo (load-path))`. See [Section 7.2.113 \[set-load-path\]](#), page 131.

7.2.64 look

`(look [objectID] [cameraID])`

Rotates the named camera to point toward the center of the bounding box of the named objeto (or the origin in hyperbolic or spherical space). In Euclidean space, moves the camera forward or backward until the objeto 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`.

7.2.65 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 7.2.66 \[look-encompass-size\]](#), page 122.

7.2.66 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 janela de camera filled by the objeto, `clip-ratio` is the max allowed ratio of near-to-far clipping planes. The near clipping plane is $1/\text{near-margin}$ times closer than the near edge of the objeto, and the far clipping plane is `far-margin` times further away. Returns the list of current values. Defaults: .75 100 0.1 4.0

7.2.67 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-Euclidean 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.

7.2.68 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.

7.2.69 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.

7.2.70 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.

7.2.71 merge-base-ap

(merge-base-ap APPEARANCE)

merge-base-ap is a synonym for [Section 7.2.72 \[merge-baseap\]](#), page 123.

7.2.72 merge-baseap

(merge-baseap APPEARANCE)

Merge in some appearance characteristics to the base default appearance (applied to every geom before its own appearance). Lighting is typically included in the base appearance.

7.2.73 morehelp

(morehelp command)

command may include * wildcards. Prints more info than [Section 7.2.54 \[help\]](#), page 120.

7.2.74 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 objeto's id thereafter.

7.2.75 ND-axes

(ND-axes CAMID [CLUSTERNAME [Xindex Yindex Zindex [Windex]]])

In our model for N-D viewing (enabled by (dimension)), objetos in N-space are viewed by N-dimensional *camera clusters*. Each real janela de câmera belongs to some cluster, and shows & manipulates a 3-D axis-aligned projected subspace of the N-space seen by its cluster. Moving one câmera 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.

7.2.76 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 objetos 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 $v = P \cdot x$ 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, objetos tagged with the `keepcolor` appearance attribute are shown in their natural colors.

7.2.77 ND-xform

(ND-xform OBJID [ntransform { idim odim ... }])

Concatenate the given ND-transform with the current ND-transform of the objeto (apply the ND-transform to objeto 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.

7.2.78 ND-xform-get

(ND-xform-get ID [from-ID])

Returns the N-D transform of the given objeto in the coordinate system of from-ID (default `universe`), in the sense $\langle \text{point-in-ID-coords} \rangle * \text{Transform} = \langle \text{point-in-from-ID-coords} \rangle$. Note that ND-transforms have their homogeneous coordinate at index 0, while 3D transform have it at index 3.

7.2.79 ND-xform-set

(ND-xform-set OBJID [ntransform { idim odim ... }])

Sets the N-D transform of the given objeto. In dimension N, this is an $(N+1) \times (N+1)$ matrix, so in that case idim and odim are expected to be both equal to $(N+1)$. Note that all câmeras 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.

7.2.80 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 objeto 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 objetos are: they don't appear in the objeto browser and the user can't move them with the normal movimento modes.

7.2.81 new-camera

(new-camera name [CAMERA])

Create a new câmera with the given name (a string). If a câmera with that name already exists, the new objeto is given a unique name. If *CAMERA* is omitted a default câmera is used.

7.2.82 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 câmera so that it is pointing parallel to the z axis toward the center of the world.

7.2.83 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

objeto is given an empty geometry. If an objeto with that name already exists, the new objeto is given a unique name.

7.2.84 new-reset

(new-reset)

Equivalent to (progn (new-center ALLGEOMS) (new-center ALLCAMS)).

7.2.85 NeXT

(NeXT) Returns `t` if running on a NeXT, `nil` if not. A relict from ancient work-station year.

7.2.86 normalization

(normalization GEOM-ID {each|none|all|keep})

Set the normalization status of GEOM-ID.

<code>none</code>	suppresses all normalization.
<code>each</code>	normalizes the objeto'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, $\sqrt{(x_{\max}-x_{\min})^2 + (y_{\max}-y_{\min})^2 + (z_{\max}-z_{\min})^2}$, is 2.
<code>all</code>	resembles <code>each</code> , except when an objeto is changing (e.g. when its geometry is being changed by an external program). Then, <code>each</code> tightly fits the bounding box around the objeto whenever it changes and normalizes accordingly, while <code>all</code> normalizes the union of all variants of the objeto and normalizes accordingly.
<code>keep</code>	leaves the current normalization transform unchanged when the objeto changes. It may be useful to apply <code>each</code> or <code>all</code> normalization apply to the first version of a changing objeto to bring it in view, then switch to <code>keep</code> .

7.2.87 not

(not EXPR)

Evaluates EXPR; if EXPR returns a non-`nil` value, returns `nil`, if EXPR returns `nil`, return `t`.

7.2.88 or

(or EXPR1 EXPR2)

Evaluates EXPR1; if EXPR1 returns non-`nil`, return its value, if EXPR1 returns `nil`, evaluate EXPR2 and return its value.

7.2.89 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 clique).

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 objeto)

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*.

7.2.90 pick-invisible

(pick-invisible [yes|no])

Selects whether picks should be sensitive to objetos whose appearance makes them invisible; default yes. With no arguments, returns current status.

7.2.91 pickable

(pickable GEOM-ID {yes|no})

Say whether or not GEOM-ID is included in the pool of objetos that could be returned from the pick command.

7.2.92 position

(position objectID otherID)

Set the transform of objectID to that of otherID.

7.2.93 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.

7.2.94 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.

7.2.95 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.

7.2.96 quit

(quit) quit is a synonym for [Section 7.2.47 \[exit\]](#), page 119.

7.2.97 quote

(quote EXPR)

returns the symbolic lisp expression EXPR without evaluating it.

7.2.98 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.

7.2.99 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.

7.2.100 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 objetos 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 56. See [Section 7.2.52 \[hdefine\]](#), page 119.
 See [Section 7.2.53 \[hdelete\]](#), page 120.

7.2.101 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.

7.2.102 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`.

7.2.103 regtable

(regtable)

shows the registry table

7.2.104 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 browser. See [Section 7.2.39 \[emodule-sort\]](#), page 118. See [Section 7.2.34 \[emodule-define\]](#), page 117.

7.2.105 replace-geometry

(replace-geometry GEOM-ID PART-SPECIFICATION GEOMETRY)

Replace a part of the geometry for GEOM-ID.

7.2.106 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.)

7.2.107 rib-snapshot

(rib-snapshot CAM-ID [filename])

Write Renderman snapshot (in RIB format) of CAM-ID to <filename>. If no filename specified, see [Section 7.2.106 \[rib-display\]](#), page 129 for an explanation of the filename used.

7.2.108 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.

7.2.109 scene

(scene CAM-ID [GEOMETRY])

Make CAM-ID look at GEOMETRY instead of at the universe.

7.2.110 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 7.2.119 \[sleep-until\]](#), page 131. See [Section 7.2.21 \[clock\]](#), page 115.

7.2.111 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.

7.2.112 set-emodule-path

(set-emodule-path (PATH1 ... PATHN))

Sets the search path for external modules. The PATH_i 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.

7.2.113 set-load-path

(set-load-path PATH1 ... PATHN)

Sets search path for command, geometry, etc. files. 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.

7.2.114 set-motionscale

(set-motionscale X)

Set the movimento scale factor to X (default value 0.5). These commands scale their movimiento 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 movimiento scale factor set by this function. `Scaleof(frame)` measures the size of the frame objeto.

7.2.115 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.

7.2.116 sgi

(sgi) Returns `t` if running on an sgi machine, `nil` if not. A relict from the old work-station years.

7.2.117 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 `!`.

7.2.118 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.

7.2.119 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.

7.2.120 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 safer 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)

7.2.121 soft-shader

(soft-shader CAM-ID {on|off|toggle})

Select whether to use software or hardware shading in that camera.

7.2.122 space

(space {euclidean|hyperbolic|spherical})

Set the space associated with the world.

7.2.123 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 stereeyes transforms, and `merge window` or `window` to set the pixel aspect ratio & window position if needed.

7.2.124 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)) (" "))"`.

7.2.125 transform

`(transform objectID centerID frameID`

`[rotate|translate|translate-scaled|scale] x y z [bbox-center|origin] [dt`
`[smooth]])`

Apply a movimento (rotation, translation, scaling) to objeto `objectID`; that is, construct and concatenate a transformation matrix with `objectID`'s transform. The 3 IDs involved are the objeto that moves, the center of movimento, and the frame of reference in which to apply the movimiento. 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 objeto will orbit the objeto do centro. Normally `frameID`, in whose coordinate system the (mouse) movimientos 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 objeto do centro.

The next field is optional and may consist of the keyword `bbox-center` or the keyword `origin` and modifies the location of the origin of `objectID`'s co-ordinate frame: `bbox-center` temporarily translates `objectID`'s co-ordinate frame to the center of `objectID`'s bounding box. `origin` is the default (and means not to modify `objectID`'s co-ordinate frame) and `bbox-center` is only valid in Euclidean space.

The optional `dt` field allows a simple form of animation; if present, the objeto moves by just that amount during approximately `dt` seconds, then stops. If present and followed by the `smooth` keyword, the movimiento is animated with a $3t^2-2t^3$ function, so as to start and stop smoothly. If absent, the movimiento is applied immediately.

7.2.126 transform-incr

```
(transform-incr objectID centerID frameID
[rotate|translate|translate-scaled|scale] x y z [origin|bbox-center] [dt
[smooth]])
```

Apply continuing movimento: 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 objeto is moved at each time step such that its average movimento equals one instance of the movimento 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.

7.2.127 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.

7.2.128 ui-cam-focus

```
(ui-cam-focus [focus-change|mouse-cross])
```

Set the focus policy for the janelas de câmera. The default is `mouse-cross`: a câmera is made the active câmera (for interactive mouse events) when the mouse cursor crosses the window. Because this means it can become complicated to activate a specific câmera (in the context of multiple janelas de câmera) there is also the option to only change the câmera 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 câmera becomes the active câmera 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 87):

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

7.2.129 ui-center

```
(ui-center ID)
```

Set the center for user interface (i.e. mouse) controlled movimentos to objeto ID.

7.2.130 ui-center-origin

```
(ui-center-origin [origin|bbox-center])
```

Set the origin of the coordinate system of the CENTER objeto for user interface (i.e. mouse) controlled movimentos. The keyword `origin` means to use the

origin of the coordinate system of the currently selected objeto, while **bbox-center** means to use the center of the bounding box of the current objeto. 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 objeto do centro is not a *geometry*, e.g. if it is a camera. Same holds if the World is the currently selected objeto.

7.2.131 ui-emotion-program

(ui-emotion-program PROGRAM)

This is an obsolete command. Use its new equivalent [Section 7.2.34 \[emodule-define\]](#), page 117 instead.

7.2.132 ui-emotion-run

(ui-emotion-run EMODULE)

This is an obsolete command. Use its new equivalent [Section 7.2.40 \[emodule-start\]](#), page 118 instead.

7.2.133 ui-freeze

(ui-freeze {on|off})

Toggle updating user interface panels. Off by default.

7.2.134 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.

7.2.135 ui-motion

(ui-motion {inertia|constrain|own-coordinates} {on|off})

Enable or disable certain properties of mouse-controlled movimento. 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 32.

inertia Normally, moving objetos have inertia: if the mouse is still moving when the botão is released, the selected objeto continues to move. When **inertia** is off, objetos cease to move as soon as you release the mouse.

constrain

It’s sometimes handy to move an objeto in a direction aligned with a coordinate axis: exactly horizontally or vertically. Calling (ui-motion constrain on) changes the interpretation of mouse movimentos to allow this; approximately-horizontal or approximately-vertical mouse dragging becomes exactly horizontal or vertical

movimento. Note that the movimento is still along the X or Y axes of the câmera in which you move the mouse, not necessarily the objeto's own coordinate system.

own-coordinates

It's sometimes handy to move objetos 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 movimientos are interpreted that way: dragging the mouse rightward in translate mode moves the objeto in its own +X direction, and so on. May be especially useful in conjunction with the `(ui-motion constrain on)` command.

7.2.136 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	movimento controls
appearance	appearance controls
cameras	câmera 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.

7.2.137 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.

7.2.138 ui-target

(ui-target ID [yes|no])

Set the target of user actions (the selected line of the objeto albo browser) to ID. The second argument specifies whether to make ID the current objeto regardless of its type. If **no**, then ID becomes the current objeto of its type (geom or camera). The default is **yes**. This command may result in a change of modos de movimento based on target choice.

7.2.139 uninterest

(uninterest (COMMAND [args]))

Undoes the effect of an **interest** command. (COMMAND [args]) must be identical to those used in the [Section 7.2.60 \[interest\], page 121](#) command.

7.2.140 update

(update [timestep_in_seconds])

Apply each incremental movimento once. Uses timestep if it's present and nonzero; otherwise movimientos are proportional to elapsed real time.

7.2.141 update-draw

(update-draw CAM-ID [timestep_in_seconds])

Apply each incremental movimento once and then draw CAM-ID. Applies timestep seconds' worth of movimento, or uses elapsed real time if timestep is absent or zero.

7.2.142 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 7.2.15 \[camera\], page 115](#) or [Section 7.2.81 \[new-camera\], page 125](#)).

7.2.143 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.

7.2.144 write

(write {command|geometry|camera|transform|ntransform|window} 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 objeto, no transformation or appearance (geometry only) world: the objeto as positioned within the World. universe: objeto's position in universal coordinates; includes World-transform other ID: the objeto 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 7.2.22 \[command\]](#), page 115).

The ID can either be a single id or a parenthesized list of ids, like `g0` or `(g2 g1 dodec.off)`.

7.2.145 write-comments

`(write-comments FILENAME GEOMID PICKPATH)`

write OOGL COMMENT objetos 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 objeto. Descent into a complex objeto (LIST or INST) adds a new integer to the path. Traversal of simple objetos 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 objetos, which can be attached to other OOGL objetos 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.

7.2.146 write-handle

`(write-handle PREFIX FILENAME HANDLE)`

Writes the objeto underlying the given handle to *FILENAME*. This function is intended for internal debugging use only.

7.2.147 write-sexpr

`(write-sexpr FILENAME LISPOBJECT)`

Writes the given LISPOBJECT to FILENAME. This function is intended for internal debugging use only.

7.2.148 xform

`(xform ID TRANSFORM)`

Concatenate TRANSFORM with the current transform of the objeto (apply TRANSFORM to objeto ID).

7.2.149 xform-incr

(xform-incr ID TRANSFORM)

Apply continual movimento: concatenate TRANSFORM with the current transform of the objeto every refresh (set objeto ID's incremental transform to TRANSFORM).

7.2.150 xform-set

(xform-set ID TRANSFORM)

Overwrite the current objeto transform with TRANSFORM (set objeto ID's transform to TRANSFORM).

7.2.151 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 botões 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 câmera has its own model setting.

The default model is all three spaces is *Virtual*. This corresponds to the câmera 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 Euclidean 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 câmera is initially outside the unit ball. In this model, the câmera moves by Euclidean movimentos and geometry moves by hyperbolic movimentos. *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 movimentos give rise to projective transformations in the *Projective* model, and to Möbius transformations in the *Conformal* model. In both of these models the câmera moves by Euclidean movimentos.

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* botão 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 7.2.111 \[set-conformal-refine\]](#), page 130.

There are several sample hyperbolic space objetos in the `'data/geom/hyperbolic'` subdirectory of the Geomview directory. Likewise, the subdirectory `'data/geom/spherical'` contains several sample spherical space objetos.

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 objeto 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 objetos 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 objetos 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 objeto *graphics* to the file named *file*.

This package also provides the `Geomview` command which sends a 3D graphics objeto 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 objeto *graphics* to Geomview as a geom named *Mathematica*. Subsequent usage of `Geomview[graphics]` replaces the *Mathematica* objeto in Geomview with the new *graphics*.

`Geomview[name,graphics]`

Sends the 3D graphics objeto *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 objetos 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 geomview and loads the graphics objeto 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 objeto by the new objeto.

```
In[4] := Geomview[{%2,%3}]
```

```
Out[4] := {-SurfaceGraphics-, -SurfaceGraphics-}
```

This displays both objetos at once. You also can have more than one Mathematica objeto 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 141](#).

```
In[5] := Graphics3D[ {RGBColor[1,0,0], Line[{ {2,2,2},{1,1,1} }} ]}]
```

```
Out[5] := -Graphics3D-
```

```
In[6] := Geomview["myline", %5]
```

This adds 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 149). 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` objetos 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` objetos correctly that Mathematica does not handle correctly, which means that `Geomview[objeto]` sometimes works where `Show[objeto]` will give errors.

The converter supports the `Polygon`, `Line`, and `Point` graphics primitives, `RGBColor` `Graphics3D` directives, and `SurfaceGraphics` objetos 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` objeto, or a string starting with a `!` to send the output to a command. Objeto can be a `Graphics3D` objeto, a `SurfaceGraphics` objeto, 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

If Geomview is properly installed on your system according to the instructions in See [Chapter 10 \[Installation\]](#), [page 148](#), 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 tailor 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 148](#). 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 ‘\$GEOMROOT/mathematica’ subdirectory of the binary distributions, and in the ‘\$GEOMROOT/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 Mathematica 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 `'oogl2ri'`. 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 `'oogl2ri'` 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 `'oogl2ri'` 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

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

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 source-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 on 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

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 146.

10.2 Compiling and Installing the Source Code Distribution

The main reason to get the source code distribution is to look at and/or work with the source code. If you are only concerned 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 object 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` commands in any subdirectory in the tree to recompile and/or install a part of Geomview or a module.

If you want to modify the compiler 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, 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 development 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 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 113, 131

<

< 113

=

= 113

>

> 113

?

? 113

?? 113

|

|, emodule-run 113, 117

A

all 114

all camera 114

all emodule defined 114

all emodule running 114

all geometry 114

and 114

ap-override 114

B

backcolor 114

background-image 114

bbox-color 114

bbox-draw 114

C

camera 115

camera-draw 115

camera-prop 115

camera-reset 115

car 115

cdr 115

clock 115

command 115

copy 116

cursor-still 116

cursor-twitch 116

D

delete 116

dice 116

dimension 116

dither 116

draw 116

dump-handles 117

E

echo 117

emodule-clear 117

emodule-define 117

emodule-defined 117

emodule-isrunning 117

emodule-path 117

emodule-run, | 113, 117

emodule-sort 118

emodule-start 118

emodule-transmit 118

escale 118

event-keys 118

event-mode 118

event-pick 119

evert 119

exit 119

ezoom 119

F

freeze 119

G

geometry 119

geomview-version 119

H

hdefine 119

hdelete 120

help 120

hmodel 120

hsphere-draw 120

I

if 120

inhibit-warning 121

input-translator 121

interest 121

L

lines-closer	121
load	122
load-path	122
look	122
look-encompass	122
look-encompass-size	122
look-recenter	123
look-toward	123

M

merge	123
merge-ap	123
merge-base-ap	123
merge-baseap	123
morehelp	123

N

name-object	124
ND-axes	124
ND-color	124
ND-xform	124
ND-xform-get	125
ND-xform-set	125
new-alien	125
new-camera	125
new-center	125
new-geometry	125
new-reset	126
NeXT	126
normalization	126
not	126

O

or	126
----------	-----

P

pick	126
pick-invisible	127
pickable	127
position	127
position-at	128
position-toward	128
progn	128

Q

quit	128
quote	128

R

rawevent	128
----------------	-----

rawpick	128
read	128
real-id	129
redraw	129
regtable	129
rehash-emodule-path	129
replace-geometry	129
rib-display	129
rib-snapshot	130

S

scale	130
scene	130
set-clock	130
set-conformal-refine	130
set-emodule-path	130
set-load-path	131
set-motionscale	131
setenv	131
sgi	131
shell, !	113, 131
sleep-for	131
sleep-until	131
snapshot	132
soft-shader	132
space	132
stereowin	132

T

time-interests	133
transform	133
transform-incr	134
transform-set	134

U

ui-cam-focus	134
ui-center	134
ui-center-origin	134
ui-emotion-program	135
ui-emotion-run	135
ui-freeze	135
ui-html-browser	135
ui-motion	135
ui-panel	136
ui-pdf-viewer	136
ui-target	137
uninterest	137
update	137
update-draw	137

W

window	137
winenter	137
write	137

`write-comments`..... 138
`write-handle`..... 138
`write-sexpr` 138

X

`xform`..... 138

`xform-incr` 139
`xform-set`..... 139

Z

`zoom`..... 139

Table of Contents

Introduction to Geomview	2
Distribution	3
Copying	4
GNU LESSER PUBLIC LICENSE	5
Introdução	5
TERMOS E CONDIÇÕES PARA CÓPIA, DISTRIBUIÇÃO E	
MODIFICAÇÃO	7
Como Aplicar Estes Termos para Suas Novas Bibliotecas.....	14
History of Geomview’s Development	15
Authors.....	15
Supported Platforms.....	16
How to Pronounce “Geomview“	17
1 Overview	18
2 Tutorial	19
3 Interaction	26
3.1 Starting Geomview.....	26
3.2 Command Line Options.....	26
3.3 Basic Interaction: The Main Panel	28
3.4 Loading Objects Into Geomview.....	30
3.5 Using the Mouse to Manipulate Objects.....	32
3.5.1 Selecting a Point of Interest	36
3.6 Changing the Way Things Look.....	37
3.6.1 The Appearance Panel	37
3.6.2 The Materials Panel.....	40
3.6.3 The Lighting Panel.....	41
3.7 Cameras	42
3.8 Saving your work	45
3.9 The Commands Panel.....	48
3.10 Keyboard Shortcuts.....	49

4	OOGL File Formats	53
4.1	Conventions	53
4.1.1	Syntax Common to All OOGL File Formats	53
4.1.2	File Names	53
4.1.3	Vertices	53
4.1.4	N-dimensional Vertices	54
4.1.5	Surface normal directions	54
4.1.6	Transformation matrices	55
4.1.7	ND Transformation matrices	55
4.1.8	Binary format	55
4.1.9	Embedded objects and external-object references	56
4.1.10	Appearances	57
4.1.11	Texture Mapping	60
4.2	Object File Formats	63
4.2.1	QUAD: collection of quadrilaterals	63
4.2.2	MESH: rectangularly-connected mesh	63
4.2.3	BBOX: simple bounding boxes	64
4.2.4	Bezier Surfaces	65
4.2.5	OFF Files	66
4.2.6	VECT Files	68
4.2.7	SKEL Files	69
4.2.8	SPHERE Files	70
4.2.9	INST Files	71
4.2.9.1	INST Examples	73
4.2.10	LIST Files	73
4.2.11	TLIST Files	74
4.2.12	GROUP Files	75
4.2.13	DISCGRP Files	75
4.2.14	COMMENT Objects	75
4.3	Non-geometric objects	76
4.3.1	Appearance Objects	76
4.3.2	Image Objects	76
4.3.3	Transform Objects	80
4.3.4	ND-Transform Objects	81
4.3.5	cameras	83
4.3.6	window	85
5	Customization: ‘.geomview’ files	87

6	External Modules	88
6.1	How External Modules Interface with Geomview	88
6.2	Example 1: Simple External Module	88
6.3	Example 2: Simple External Module with FORMS Control Panel	92
6.4	The XForms Library	96
6.5	Example 3: External Module with Bi-Directional Communication	96
6.6	Example 4: Simple Tcl/Tk Module Demonstrating Picking	105
6.7	Module Installation	109
6.7.1	Private Module Installation	109
6.7.2	System Module Installation	109
7	GCL: the Geomview Command Language	111
7.1	Conventions Used In Describing Argument Types	111
7.2	GCL Reference Guide	113
7.2.1	!	113
7.2.2	<	113
7.2.3	=	113
7.2.4	>	113
7.2.5	?	113
7.2.6	??	113
7.2.7		113
7.2.8	all	114
7.2.9	and	114
7.2.10	ap-override	114
7.2.11	backcolor	114
7.2.12	background-image	114
7.2.13	bbox-color	114
7.2.14	bbox-draw	114
7.2.15	camera	115
7.2.16	camera-draw	115
7.2.17	camera-prop	115
7.2.18	camera-reset	115
7.2.19	car	115
7.2.20	cdr	115
7.2.21	clock	115
7.2.22	command	115
7.2.23	copy	116
7.2.24	cursor-still	116
7.2.25	cursor-twitch	116
7.2.26	delete	116
7.2.27	dice	116
7.2.28	dimension	116
7.2.29	dither	116
7.2.30	draw	116
7.2.31	dump-handles	117

7.2.32	echo	117
7.2.33	emodule-clear	117
7.2.34	emodule-define	117
7.2.35	emodule-defined	117
7.2.36	emodule-isrunning	117
7.2.37	emodule-path	117
7.2.38	emodule-run	117
7.2.39	emodule-sort	118
7.2.40	emodule-start	118
7.2.41	emodule-transmit	118
7.2.42	escale	118
7.2.43	event-keys	118
7.2.44	event-mode	118
7.2.45	event-pick	119
7.2.46	evert	119
7.2.47	exit	119
7.2.48	ezoom	119
7.2.49	freeze	119
7.2.50	geometry	119
7.2.51	geomview-version	119
7.2.52	hdefine	119
7.2.53	hdelete	120
7.2.54	help	120
7.2.55	hmodel	120
7.2.56	hsphere-draw	120
7.2.57	if	120
7.2.58	inhibit-warning	121
7.2.59	input-translator	121
7.2.60	interest	121
7.2.61	lines-closer	121
7.2.62	load	122
7.2.63	load-path	122
7.2.64	look	122
7.2.65	look-encompass	122
7.2.66	look-encompass-size	122
7.2.67	look-recenter	123
7.2.68	look-toward	123
7.2.69	merge	123
7.2.70	merge-ap	123
7.2.71	merge-base-ap	123
7.2.72	merge-baseap	123
7.2.73	morehelp	123
7.2.74	name-object	124
7.2.75	ND-axes	124
7.2.76	ND-color	124
7.2.77	ND-xform	124
7.2.78	ND-xform-get	125
7.2.79	ND-xform-set	125

7.2.80	new-alien	125
7.2.81	new-camera	125
7.2.82	new-center	125
7.2.83	new-geometry	125
7.2.84	new-reset	126
7.2.85	NeXT	126
7.2.86	normalization	126
7.2.87	not	126
7.2.88	or	126
7.2.89	pick	126
7.2.90	pick-invisible	127
7.2.91	pickable	127
7.2.92	position	127
7.2.93	position-at	128
7.2.94	position-toward	128
7.2.95	progn	128
7.2.96	quit	128
7.2.97	quote	128
7.2.98	rawevent	128
7.2.99	rawpick	128
7.2.100	read	128
7.2.101	real-id	129
7.2.102	redraw	129
7.2.103	regtable	129
7.2.104	rehash-emodule-path	129
7.2.105	replace-geometry	129
7.2.106	rib-display	129
7.2.107	rib-snapshot	130
7.2.108	scale	130
7.2.109	scene	130
7.2.110	set-clock	130
7.2.111	set-conformal-refine	130
7.2.112	set-emodule-path	130
7.2.113	set-load-path	131
7.2.114	set-motionscale	131
7.2.115	setenv	131
7.2.116	sgi	131
7.2.117	shell	131
7.2.118	sleep-for	131
7.2.119	sleep-until	131
7.2.120	snapshot	132
7.2.121	soft-shader	132
7.2.122	space	132
7.2.123	stereowin	132
7.2.124	time-interests	133
7.2.125	transform	133
7.2.126	transform-incr	134
7.2.127	transform-set	134

7.2.128	ui-cam-focus	134
7.2.129	ui-center	134
7.2.130	ui-center-origin	134
7.2.131	ui-emotion-program	135
7.2.132	ui-emotion-run	135
7.2.133	ui-freeze	135
7.2.134	ui-html-browser	135
7.2.135	ui-motion	135
7.2.136	ui-panel	136
7.2.137	ui-pdf-viewer	136
7.2.138	ui-target	137
7.2.139	uninterest	137
7.2.140	update	137
7.2.141	update-draw	137
7.2.142	window	137
7.2.143	winenter	137
7.2.144	write	137
7.2.145	write-comments	138
7.2.146	write-handle	138
7.2.147	write-sexpr	138
7.2.148	xform	138
7.2.149	xform-incr	139
7.2.150	xform-set	139
7.2.151	zoom	139
8	Non-Euclidean Geometry	140
9	Mathematica Graphics in Geomview or RenderMan	141
9.1	Using Mathematica to generate OOGL files	141
9.2	Using Geomview as Mathematica's Default 3D Display	142
9.3	Using Mathematica to generate RenderMan files	143
9.4	Using Geomview and Mathematica on Different Computers ...	144
9.4.1	Using a Networked Geomview Host	144
9.4.2	Transporting Mathematica Files to Geomview by Hand ..	145
9.5	Details of the Mathematica->Geomview Package	146
9.6	Installing the Mathematica Packages	146
10	Installation	148
10.1	Installing the Unix Binary Distribution	148
10.1.1	Details of the Unix Binary Installation	149
10.2	Compiling and Installing the Source Code Distribution	149
	Getting Technical Support for Geomview	151
	Contributing to Geomview's Development	152

Function Index	153
-----------------------------	------------

List of Figures

Figure 2.1: Initial Geomview display.	19
Figure 2.2: Olhando para o Mundo.	21
Figure 2.3: A Paine! Aparência.	22
Figure 2.4: Paine! de Escolha de Cores.	22
Figure 2.5: O Paine! de Arquivos.	24
Figure 2.6: Trevo e Dodecaedro.	25
Figure 3.1: The Main Panel.	28
Figure 3.2: O Paine! de Arquivos.	31
Figure 3.3: The Load Panel.	32
Figure 3.4: O Paine! de Ferramentas.	33
Figure 3.5: The Appearance Panel.	38
Figure 3.6: Color Chooser Panel.	39
Figure 3.7: The Materials Panel.	40
Figure 3.8: The Cameras Panel.	43
Figure 3.9: The Save Panel.	46
Figure 3.10: The Commands Panel.	48