A Guide to Using `geomview` and *OOGL*

Charlie Gunn

Revision: 1.3
Date: 1992/09/22 20:54:26

1 Introduction

This document will show you how to write programs using the Object Oriented Graphics Library (*OOGL*). It will also give you an introduction to using **geomview**—a 3-dimensional viewing program written using *OOGL*. By using **geomview**, your application is freed from having to do any specific graphics functions. [1]

The *OOGL* routines are a high-level set of modules that let you specify and display geometry. There are several types of geometric primitives which OOGL can handle, including polygons, vectors, meshes, and parametric surfaces patches. In addition, these primitives can be organized into a higher-order lists. Also, multiple instances of the same geometry can be created by specifying different modeling transformations. Each sort of geometric object, whether low-level or high-level, comes equipped with a standard set of operations. Examples of these operations include loading from and saving to files, drawing to a display device, and computing a bounding box. This style of programming, which combines data structures with operations which act on these data structures, is known as object-oriented programming. Although currently *OOGL* only works on SGI workstations, it has been designed to make it easy to port to other environments.

It is assumed you know how to program in C. After reading this and looking through the example programs you will be ready to create your own *OOGL* applications.

[1] References to files in this document will use the symbol GEOM to refer to the directory where the geomview distribution has been installed on your system.

2 Integrating your application and and `geomview`

2.1 Why Use `geomview`?

`geomview` is a program for the display and manipulation of *OOGL* objects. If you haven't already done so, you should familiarize yourself with its capabililities now. Consult the documents `overview` and `oogltour` in the `doc` directory, and the man page for `geomview`, both sections 1 and 5. The section 1 manual page describes the user interface of the program, while the section 5 manual page describes the command language through which external programs can communicate with `geomview`. It is this command language which this example will use.

This corresponds to 2 different ways to provide `geomview` with geometry for viewing. The first reads descriptions of objects from a file. `geomview` has a file browser from which users can choose files to examine. This is appropriate when the geometry is static. The second method is useful when you desire to view a changing geometry. This second method uses 2-way pipes to update the changing geometry from your application and to pass back information from `geomview` to your application. This communication uses the command language described in `geomview(5)`. In this situation, your application is known as an *external module* for `geomview`.

This document will show how to create and run an external module, in this case one simply known as `example`. This and all the other files involved in this are to be found in the directory `GEOM/src/bin/example`.

3 How to Write an External Module

3.1 Structure of the *OOGL* Routines

OOGL stands for "Object-Oriented Graphics Library." The library defines a class of "geometry objects." As indicated above, these are object-oriented in the sense that a geometry object is a data structure plus a set of procedures, often called "methods", which operate on the data. The object is accessible only through its methods.

Another feature of object-oriented programming is the capacity to create subclasses of a class. There are a number of subclasses of the generic geometry class Geom. In practice, each instance of the Geom class is also an instance of a specific subclass, but this isn't necessarily true in general. Example subclasses include: vectors, polygons, meshes, parametric surface patches, and bounding boxes. There are also composite subclasses built on these, such as lists and instances. See oogl(5) for a description of the file formats belonging to these subclasses.

All subclasses of geometry objects have a set of standard methods used to manipulate the objects. Some example methods are: Create, Delete, Draw, Load, Save, and Print. The name of a subclass's methods begin with the name of that subclass. For example, the Draw method for a mesh is named MeshDraw. In general a subclass can inherit methods from its superclasses; but in *OOGL* almost all subclass methods are provided specifically for that subclass. These methods behave differently because they are each customized to the particular subclass of geometry object to which they belong. The details of the data structures and existing methods for each subclass can be found in the include file for that subclass. These are kept in `GEOM/include`.

3.1.1 The Generic Class Geom

There is a type of programming task which is made much easier by the fact that each instance of a subclass is also an instance of the generic class. In our case, each instance of a geometry subclass is also an instance of the generic Geom class. So if our program, like `geomview`, is designed to deal with many different types of geometry objects, we can simplify the program structure by always working with the Geom class rather than the specific subclasses. GeomLoad and GeomDraw, for example, work perfectly well to load and draw instances of any subclass, too. Of course it takes some work to make sure this works, but that work has already been done for you.

3.2 `example`: A Minimal *OOGL* Program

`example` demonstrates the use of the mesh *OOGL* object. It repeatedly computes a changing function on a rectangular grid, or mesh, of (x, y) pairs. As the mesh is changed by `example` over time, these changes dynamically appear in the `geomview` window.

A second feature of `example` is the use of the Forms interface builder to create the user interface. This is a public domain interface builder which has been used to create `geomview`. It is included here because we have found it to be an effective tool for creating interactive applications. The Forms package is written by Mark Overmars of Utrecht University and is available via anonymous ftp from archive.cs.ruu.nl.

3.2.1 Preparing to run `example`

First, check that `GEOM/bin/sgi/example` exists. If not, first change directory to `GEOM/src/bin/example` and type

```
% make install
```

You'll next need to be sure that `example` is registered with `geomview`. To do this, edit the file `.geomview` in your home directory and add the line

```
(emodule-define "Example" "example")
```

This file will be read by **geomview** when it starts up and the line you've added tells it to register the external module **example** to its list of external modules, under the name "Example". As long as **example** exists in **GEOM/bin/sgi**, **geomview** will find it there. Or, you can provide a full pathname for your application, which will guarantee that it will be found. Also, if your application has command line arguments, include them here.

Then, run **geomview**. On the main panel, under the "Modules", "Example" should appear. Pick this entry. It should be highlighted in yellow, and you should be prompted to place a small panel on the screen. This is the user-interface panel create using the Forms designer. Soon after you place this panel, you should see an object appear in the geomview viewing window. This is the output from the **example** module. As you watch, the geometry should change. If you want this change to be slower or faster, you can move the single slider in the panel which controls the 'velocity' of the change.

3.2.2 The `example` Code

The file **main.c** contains all of the non-*OOGL* code. The routine **myfunc** is the function to be computed over the mesh. Its arguments are the x and y location on the mesh and the time since the start of the program.

The main routine first performs a number of initialization steps. The call to `Begin_OOGL` essentially creates an empty geometry template and returns:

```
fprintf(f, "(geometry example { : exhandle })\n");
fflush(f);
```

This notifies the viewer to create an object known as *example*. It gives it the internal name, or handle, *exhandle*. More on this later when we come to `UpdateOOGL`

After initializing everything, **main** enters an endless loop where it computes the values of the function over each point in the mesh and then calls `UpdateOOGL` to update the *OOGL* mesh in shared memory. User interface allows the user to change the value of dt. The resulting data array and other parameters are passed to `UpdateOOGL`.

`UpdateOOGL`'s main task is to create a Mesh object from the data which has been passed in. This is the one slightly tricky section of the program, since the particular features of each subclass must be known in order to create an instance.

All the Create routines are accessed through the generic `GeomCreate` routine. The first parameter is an ascii string identifying the subclass, in this case "mesh". From then on, the Create routine expects a sequence of keys accompanied with optional key values. A complete list of these keys is given in `GEOM/include/create.h`. There are a number of generic sorts of keys, such as CR_NOCOPY or CR_POINT, which are valid for any subclass. CR_NOCOPY tells the create routines to use the data you have provided without copying it over – so we are careful not to delete the data storage we are passing. CR_POINT identifies an array of 3-dimensional vertices; CR_POINT4 does the same for 4-dimensional vertices.

Other of these keys, however, are specific to the Mesh subclass. For example, CR_NU and CR_NV specify the dimensions of the mesh. Last and most difficult, CR_FLAG identifies a set of flag values which vary with each subclass and can only be fully learned by consulting the specific subclass.h file. In this case, we consult `meshflag.h` to find that we must use the MESH_Z flag for our data, since we aren't providing explicit (x,y) data with the z-values.

The best way to become comfortable with this aspect of the Create routines is to examine other examples in the `GEOM/src/bin` directory, and explore the include files corresponding to your favorite subclass.

`GeomCreate` returns a pointer to a Geom upon successful completion. It also happens to be a pointer to a Mesh, but we don't care any more what kind of Geom it is.

The final step is to put this mesh out onto the pipe. We do this by wrapping a call to `GeomFSave` within a simple command which connects this geometry to the afore-mentioned handle *exhandle*. `GeomFSave` prints onto the given file stream, in this case, standard output, the value of the Geom. Then, when `geomview` reads this stream, it will redefine *exhandle* to replace any old definitions.

4 Remarks

It is a good idea to use the `OOGLNew` routines and their variants for allocating storage and `OOGLFree` for freeing it. See `GEOM/include/ooglutil.h` for details.

A good way to debug external modules is to run them stand-alone first, and examine the ascii stream which they produce to check for obvious problems. Or catch it this stream in a file, edit the file, and then read that into `geomview` to see if it's making reasonable pictures.

The Makefile mechanism illustrated in this example is quite complex. The basic structure is that the source code exists at the `GEOM/src/bin` level, but there are multiple object directories corresponding to possibly different architectures (in our case `O.sgi` is the one of interest). The parent Makefile then descends into that directory and does the heart of the make. All the Makefiles include global makefiles from `GEOM/makefiles`, which define many macros that you don't need to understand. The important ones are explained below. There is a file called `Makedefs` in the parent directory that defines the macros SRCS, OBJS, and TARGET, which you will need to set to the appropriate filenames. (The DISTFILES macro can be safely ignored.) The only change you should need to make to the `Makefile` in the parent directory is to set the GEOM variable to the appropriate directory, which is the top of the geomview tree. This change also should be made to `O.sgi/Makefile`, which you can then modify as appropriate for your application. Remember that after modifying any of the makefiles you should type

```
% make depend
```

Enjoy.

5 Listings

These are the listings of all the demo code mentioned in this article.

5.1 example

5.1.1 File: main.c

```
/*
 *      file:    main.c:
 *      author: Charlie Gunn & Tamara Munzner
 *      date:    September 1, 1992
 *
 *      simple example of geomview external module and
 *    OOGL graphics library routines.
 *
 *      The main program continually computes a function on a mesh of
 *      (x,y) pairs.   The updated mesh is printed to stdout.  When this
 *    program is invoked as a geomview external module, pipes are hooked
 *     up
 */

#include <math.h>
#include <stdio.h>
#include "forms.h"
#include "panel.h"
#include "ooglutil.h"

float dt;


/* replace this with your favorite function */
float
myfunc(x,y,t)
    float x,y,t;
{
        float r;
```

```
        r = sqrt(x*x+y*y) + .000001;
        return(sin(r + t)*sqrt(r));
}

main(argc, argv)
     char **argv;
{
        int xdim, ydim, i, j;
        float xmin, xmax, ymin, ymax, xsize, ysize, dx, dy, x, y, t, zscale;
        float *data;

        xdim = 24;
        ydim = 24;
        xmin = -5;
        xmax = 5;
        ymin = -5;
        ymax = 5;
        zscale = 2.0;

    dt = .1; /* initial velocity */

    /* geomview communications setup. */

        Begin_OOGL();

    /* If we don't foreground then the process forks and dies
       as soon as we do graphics. This is bad.
     */

    foreground();

    /* This routine is defined in the code generated by
       the forms designer.
     */

        create_the_forms();

    /* We set the slider and display the form. */

    fl_set_slider_bounds(VelocitySlider, 0.0, 1.0);
    fl_set_slider_value(VelocitySlider, dt);
    fl_show_form(Example, FL_PLACE_SIZE, TRUE, "Example");
```

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

    data = (float *) OOGLNewN(float, xdim * ydim);
    for (t=0; ; t += dt)
    {
      /* Let forms library do its thing. */
      fl_check_forms();

        /* compute mesh of some function value */
        for (j=0, y = -ysize/2; j<ydim; ++j, y += dy)
        {
            for (i=0, x = -xsize/2; i<xdim; ++i, x += dx)
            {
                data[j*xdim + i] = myfunc(x,y,t);
            }
        }

        /* geomview communications update */
        UpdateOOGL(xdim, ydim, zscale, data);
    }

}
```

5.1.2 File: `oogl.c`

```
/* oogl.c */
/* geomview communication code */
/* Charlie Gunn & Tamara Munzner */
/* 9/92 */

#include <stdio.h>
#include "geom.h"
#include "meshflag.h"

FILE *f = stdout;

Begin_OOGL()
{

 fprintf(f, "(geometry example { : exhandle })\n");
 fflush(f);
```

```
}

UpdateOOGL(x_size, y_size, gridunit, data)
    int x_size, y_size;
    float gridunit;
    float data[];
{
        register int    x,y,k;
        Point3  *points;
        Geom *mesh;

   points = OOGLNewN(Point3, x_size*y_size);

        for (k = 0, y=0; y<y_size; ++y)
        {
                for (x=0; x<x_size; ++x, ++k)
                {
                        points[k].x = x*gridunit;
                        points[k].y = y*gridunit;
                        points[k].z = data[k];
                }
        }
        mesh = GeomCreate("mesh",
              CR_NOCOPY, /* don't copy the points */
              CR_FLAG, MESH_Z,
              CR_NU, x_size,
              CR_NV, y_size,
              CR_POINT, points,
              CR_END);

   fprintf(f, "(read geometry { define exhandle \n");
   GeomFSave(mesh, f, NULL);
   fprintf(f, "})\n");
        OOGLFree(mesh);
}
```

5.1.3 File: Makedefs

```
SRCS = main.c oogl.c callbacks.c
OBJS = main.o oogl.o callbacks.o
TARGETS = example
DISTFILES = panel.fd panel.c tutorial.tex tutorial.ps README
```

5.1.4 File: `Makefile`

```
GEOM = ../../..
include ${GEOM}/makefiles/Makedefs.global
include Makedefs
include ${GEOM}/makefiles/Makerules.src
```

5.1.5 File: `O.sgi/Makefile`

```
GEOM = ../../../..
include ${GEOM}/makefiles/Makedefs.global
include ../Makedefs
include ${GEOM}/makefiles/Makerules.obj

FORMSLIBS = -lforms -lfm_s
ALLLIBS = ${ALLOOGLLIBS} ${FORMSLIBS} -lgl_s -lm

example: ${OBJS}
rm -f example ../example
cc ${CFLAGS} ${OBJS}  ${ALLLIBS} -o example
ln example ..

tutorial.dvi: tutorial.tex
latex tutorial

install: install_bin

install_bin: example
${INSTALL} -O -v -F ${GEOM}/bin/${MACHTYPE} example
strip ${GEOM}/bin/${MACHTYPE}/example
chmod 555 ${GEOM}/bin/${MACHTYPE}/example
```