NORTHWESTERN UNIVERSITY

THE PROTOTYPING OF GEOMANAGER: A GEOMETRIC ALGORITHM MANIPULATION SYSTEM

A PROJECT REPORT

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Computer Science

By
KIYOKO F. AOKI
EVANSTON, ILLINOIS
December 1995

Acknowledgments

I would not have been able to complete this project without the support of the following people, to whom I owe much gratitude. My deepest appreciation goes to:

♡ my advisor, Prof. D.T. Lee, for his guidance that has helped me achieve more than I thought I could, and for his continuous desire to help students develop their full potential.

♡ Professors S. Kumar and E. Schwabe for their assistance any time I have asked for it, and for taking the time to serve on the examination committee.

♡ my fellow GeoMAMOS project team members and friends, especially Chin-Fang (Cindy) Shen, Dennis Sheu, and Laurent Lin, for all of their help and patience in teaching me what I needed to know (and more) to get to this point.

♡ Soka Gakkai International, the organization without which I would have never been able to keep myself together through the hardest of times.

♡ my siblings and friends, for their friendship, advice and for just being there.

♡ Shin Kurokawa, for his consideration and support for everything I work for.

♡ my parents, Takayuki and Kazuko Aoki, for their love and encouragement, even on those days when I did not return home because I stayed in the lab all night.

Contents

List of Figures

Chapter 1

Introduction

1.1 Description of GeoMAMOS

GeoMAMOS, which stands for Geometric Manipulation and Monitoring System, is a visualization software tool package, composed of several components that provide different tools for visualization and debugging C/C++ code that emulates geometric algorithms. These components consist of GeoPanel, GeoGDB, GeoSheet, and GeoLIB (see Figure 1).

GeoPanel is a graphical user interface which provides access to all of the other components. GeoGDB is the geometric algorithm debugger based on xxgdb, GeoSheet is the graphical interface for entering and modifying geometric data, and GeoLIB is a C/C++ library of geometric data types, routines and algorithms. The user programs may either use the I/O (reading and writing objects to and from GeoSheet) and geometric functions (intersection of two segments, for example) to develop new algorithms, or they may include an algorithm function provided by GeoLIB. In either case, they must first set up a connection with GeoIPC, the communication link between all of the GeoMAMOS components. A user's program may be converted to an algorithm function simply by taking out the GeoIPC setup function and passing the objects to the program as its parameters instead of explicitly reading them in one at a time within the program. Thus different objects may be passed to the same function to run the same algorithm, and the final object may be the output returned from the algorithm function. Compiling this new algorithm function into GeoLIB will then make it available to other programs.

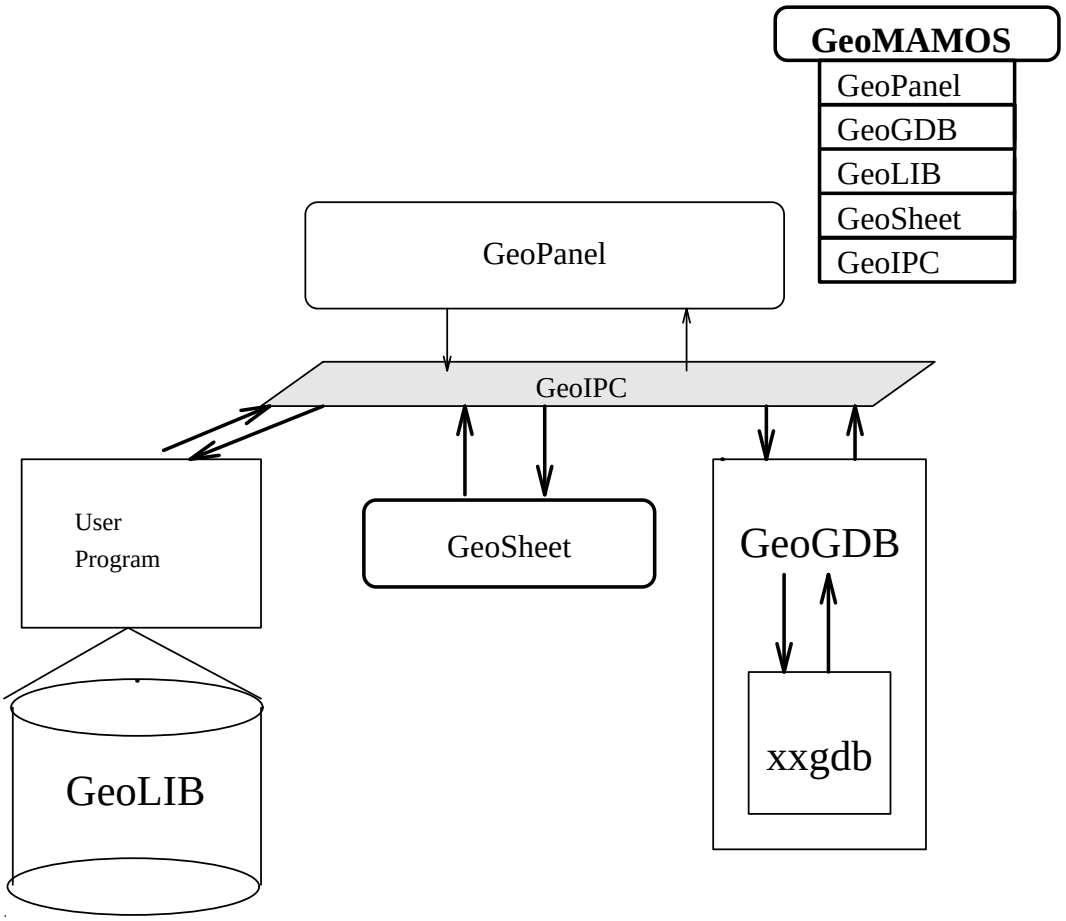

Figure 1: GeoMAMOS and its components.

1.2 Description of GeoSheet and GeoIPC

The components of GeoMAMOS that this project focused on are GeoSheet and GeoIPC. GeoSheet is the main input/output device that visualizes the user's algorithm code. It provides high-level graphical input/output manipulation facilities for geometric objects, and can be invoked from either the GeoPanel or by the user.

GeoIPC is the backbone of GeoMAMOS. It is the interprocess communication link between GeoMAMOS and all of its components, and between the user's program and GeoSheet. It is based on the transmission of UDP packets between UNIX sockets. This link, however, is transparent to the user, as the user is only required to include an `IPCServiceSetup()` function at the start of each of their programs, which basically sets up the socket by which it can communicate with one or more of GeoMAMOS'

components. The rest of the program would only consist of `Graphic_Read()` and `Graphic_Write()` functions that read and write data from and to GeoSheet through IPC.

1.3 Description of the Project

1.3.1 Limitations of the original configuration

In the original configuration, the user develops code that is visualized on GeoSheet. However, the algorithm can be visualized only once for every program execution. In order to experiment with various inputs to the algorithm, the user must either re-execute the program for each input to see how the corresponding output is affected, or explicitly include a loop in their program to run their algorithm repeatedly, during which the various inputs would be re-entered for every execution.

If the user wished to use more than one GeoSheet for their algorithm, the program would also have to call for exactly however many GeoSheets it expected to use. So in a situation where the output for an algorithm placed on one GeoSheet is to be displayed on numerous GeoSheets at different sites, the program would have to set up a connection and explicitly call `Graphic_Write()` to display the output for each of those GeoSheets for which it had a connection.

1.3.2 The solution

In order to overcome these limitations, this project will give GeoMAMOS the functionality that will enable the user to:

1. use the mouse to move the input objects to their algorithm and watch their algorithm, being executed repeatedly, dynamically update its output, and

2. simultaneously "duplicate" the execution of a program on multiple GeoSheets without having to hard-code the number of GeoSheets to use.

1.3.3 Implementation approaches

There are various approaches one may take to implement the mouse-related portion of this project. One approach would be to have the user manage the mouse-related functions, where the user program would loop the execution of their algorithm repeatedly while the objects are being moved. However, this approach would make the user be responsible for taking care of all of the functions relevant to the mouse, thus

making GeoSheet lose some of its user transparency.

Another approach would be to modify GeoSheet so that the repeated executions of the algorithm could be done from within GeoSheet itself. A pull-down menu of all of the algorithms in GeoLIB would be made available to the user for execution on the current objects on GeoSheet. However, in order to implement this, the code for GeoSheet would have to be compiled with GeoLIB, which is an unfavorable approach, as the modularity of the current system would be lost.

A third approach would be to create a new application that acts as an interface between the user program and GeoSheet. This would allow the current libraries to remain intact and maintain the modular style of GeoMAMOS. It would also keep the mouse-related functions transparent to the user. Additional advantages to this approach would be that by having this new "layer" keep a copy of the objects being used between the user program and GeoSheet, it can make sure that they remain consistent, which had been a problem in the past. This would also solve the multiple-GeoSheet problem, since this "layer" can simply duplicate the messages it is sending to all of the GeoSheets. This new "layer" approach, however, would increase the number of packets being transmitted in the system since messages would be sent between the user program and the new layer and GeoSheet, instead of only between the user program and GeoSheet.

Considering the amount of changes required in the current system among all of these approaches, however, this last option would be the most feasible. The consequences of the increase in the number of packets would have to be evaluated as this project is developed. Because it has the potential to not only give dynamic functionality to GeoSheet but also provide a managerial entity to the entire system, this project has become known as the GeoManager project.

Chapter 2

Project Requirements

In any project, there are specifications as to what the project should do and what limitations are set for the implementation of the project. This chapter will briefly explain the three main requirements for the GeoManager Project.

2.1 Transparency to the user

The first requirement for this project is that the user should not be aware of the IPC link between each of the components. Currently GeoMAMOS is set up in this manner. The user program is required only to include a setup function at the beginning and to use the objects and functions provided by GeoLIB in order to interact with GeoMAMOS. This transparency should be maintained with the addition of GeoManager.

2.2 Hard-coded algorithms in GeoSheet

One of the approaches considered for this project was to include a list of geometric algorithms that could be easily executed from GeoSheet. Currently, there is an "algorithm browser" on GeoSheet which basically displays and runs user programs that each run one algorithm. With this browser, however, there can be no pipelining between algorithms, where the output of one algorithm could be used for another, without coding this into one program.

The only other way to enable pipelining of algorithms would be to include a listing of all of the available algorithm functions in GeoLIB. However, this should be done without having to compile GeoLIB into GeoSheet. Therefore, in order to maintain the

modularity of GeoMAMOS, the second requirement for this project is that algorithm functions should not be hard-coded into GeoSheet.

2.3 System independence

The third requirement is that the final product should be system-independent, implying that the low-level mouse-related functions used by the graphic editor on which GeoSheet is based should not be modified. GeoManager should be implemented in a way that keeps the original system intact. In this way, portability can be maintained, as the current system has the potential to be ported to use other graphic editors as a base for GeoSheet as well as to use other operating systems on which GeoMAMOS could run.

Chapter 3

Project Implementation

This chapter covers the components of GeoMAMOS that were modified in order to implement the GeoManager project.

3.1 Function pointers

In order to maintain a pointer to the algorithm function being executed in the user's program, a global function pointer had to be created. However, to keep this fact transparent to the user, just as `Graphic_Read()` and `Graphic_Write()` are used to read and write data from and to GeoSheet, a `Run_alg(&alg, geolist)` function was created, where `&alg` is a pointer to a geometric algorithm function and geolist is a list of pointers to the input data objects. Calling `Run_alg(&alg, geolist)` will execute the specified algorithm function with the corresponding parameter objects.

At this point, however, there was no way to maintain a generic list of Geo-Objects. So a new object class called GSArguments was created, from which all of the current Geo-Objects could be inherited. This class would be used primarily for the `Run_alg(&alg, geolist)` function.

3.2 GSArguments

In order to be able to recognize the data type of the derived geometric class, the GSArguments class contains a pointer for each type of object defined in GeoLIB, along with an ArgType variable indicating the type of the object.

The following is the code for the GSArguments class:

```
class GSArguments  {
```

```
public:
        virtual ~GSArguments();
        GeoPoint* GetGeoPoint() { return GSPoint; };
        geopolygon* Getgeopolygon() { return GSPolygon; };
/* Get- functions for all other object types */
        virtual void SetObject(GeoPoint &p) {
                ArgType = p.GetGSObjectType();
                GSPoint = new GeoPoint(p.xcoord(), p.ycoord());};
        virtual void SetObject(geopolygon *p) {
                ArgType = p->GetGSObjectType();
                GSPolygon = new geopolygon;
                GSPolygon = p;};
/* SetObject function for all other data object types */
        GSArguments() {
                ArgType = 0; };
        Graphic_Write() {
                switch(ArgType) {
                   case GEO_DT_POINT:
                        GSPoint->Graphic_Write();
                        break;
                   case GEO_DT_POLYGON:
                        GSPolygon->Graphic_Write();
                        break;
   /*  ... other data class types ... */
                   default: break;
                }
        };
        Graphic_Read() {
                switch(ArgType) {
                   case GEO_DT_POINT:
                        GSPoint->Graphic_Read();
                        break;
                   case GEO_DT_POLYGON:
                        GSPolygon->Graphic_Read();
                        break;
   /*  ... other data class types ... */
                   default: break;
                }
        };
```

```
private:
        int ArgType;
        GeoPoint *GSPoint;
        geopolygon *GSPolygon;
/* ... other data object pointers ... */
};
```

An array of pointers to these GSArguments objects can then be used to pass to the `Run_alg(&alg, geolist)` function as `geolist`. `Run_alg(&alg, geolist)` would then pass the argument list to the function pointed to by `&alg`. To illustrate, the code for `Run_alg(&alg, geolist)` is as follows:

```
GSArguments* Run_alg(GSArguments* (*alg1)(GSArguments **gsargslist),
     GSArguments **gsalist1)
{

        GSArguments *final;

        algname = alg1;
        GSInputList = gsalist1;

        final = (*algname)(GSInputList);

        return final;
}
```

`Run_alg(&alg, geolist)` saves the algorithm function pointer as well as the input and output object lists in the global variables `algname` and `GSInputList` respectively. The output returned by the function is also of type GSArguments.

3.3 GeoManager

The basic idea of GeoManager is for it to act as a server to both the user program and GeoSheet. The user program requests information from GeoManager, and GeoSheet requests confirmation of certain actions taking place on GeoSheet (i.e., GeoSheet checks with GeoManager if the user tries to delete an object that was placed on GeoSheet by a `Graphic_Write()` command from the user program). GeoSheet also acts as a server to GeoManager, since GeoManager passes the same display-related requests to GeoSheet that it receives from the user program.

The original configuration between the user program and GeoSheet is given in Figure 2, and the updated configuration including GeoManager is illustrated in Figure 3. Both of these show the direction in which requests are being made through GeoIPC.

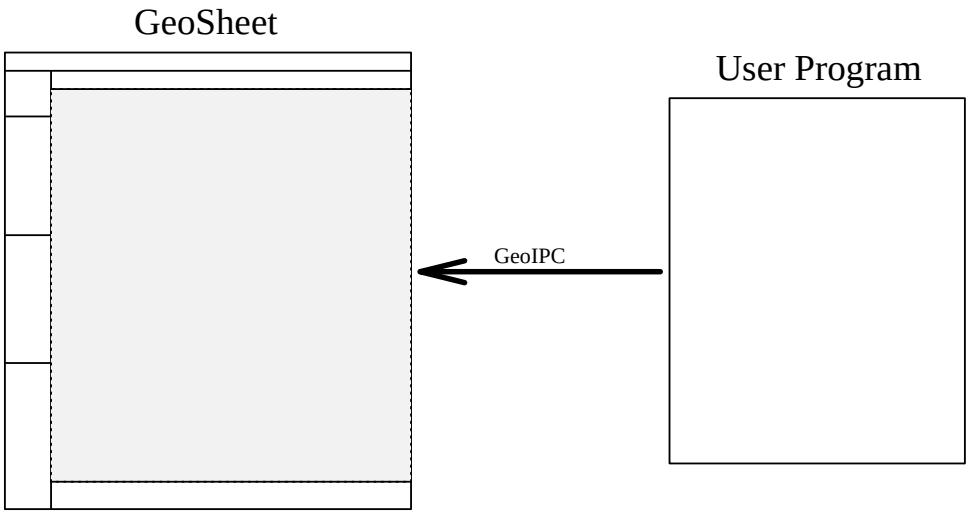

Figure 2: Original configuration between a user program and GeoSheet.

3.4 Interprocess Communication (IPC)

A service exists on a host, and is identified by its port. A message is sent to a server by sending it to the port for that service on the host on which it is running. GeoSheet creates its port with a `socket()` call. The port number chosen for GeoSheet is different every time GeoSheet is run. So in order for user programs to determine to which port to send its packets to communicate with Geosheet, after GeoSheet is started up and displays its port number, the user manually types in this port number as input to the user program as it is executed.

In order for GeoManager to avoid having its port number changed every time it is run, a port number not listed in the `/etc/services` list was chosen for it to always use, port 9876. After setting up its port using `socket()`, GeoManager then makes its existence known to the system by using the `bind()` function. In this way, the user program and GeoSheet would only have to ask for the host on which GeoManager is running, and then connect to port 9876 on that host.

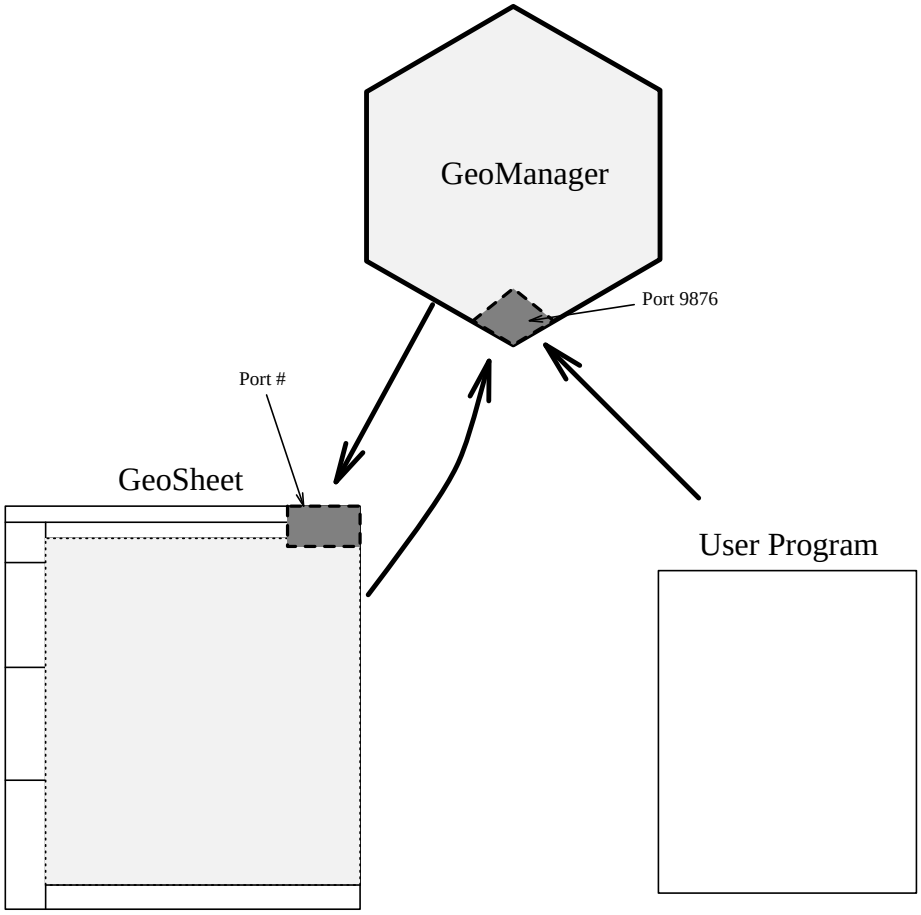

Figure 3: New configuration between a user program and GeoSheet with GeoManager.

As the server for both GeoSheet and the user program, GeoManager simply waits for messages to arrive on its port. Both GeoSheet and the user program send messages to GeoManager using the `sendto()` function, and GeoManager receives these messages with the `recvfrom()` function. When GeoManager then sends messages back to either of its clients, it uses the `sendto()` function while the clients are waiting for messages using the `recvfrom()` function. For the clients, the `sendto()` and `recvfrom()` functions are always used in pairs, as every time a request is sent, it must wait for a response, whether it is an acknowledgement or the data for which it requested. The server, on the other hand, is always waiting for messages and only uses the `sendto()` function in response to the messages it has received. A diagram of this client-server protocol is given in Figure 4.[1]

[1]Jan Newmarch. http://pandonia.canberra.edu.au/ClientServer/socket.html

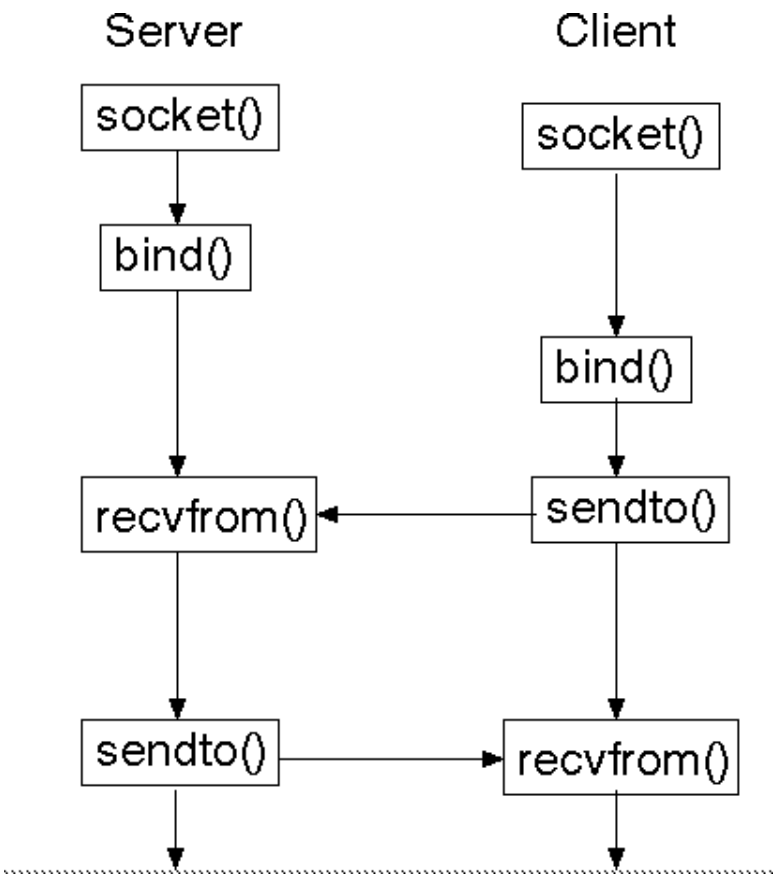

Figure 4: The UDP Protocol.

The messages that are sent between the client and server are sent in *packets*. The term *messages* and *packets* will be used interchangeably.

In implementing GeoManager, the original structures of the packet were kept the same, as changing this structure would have meant the modification of every object defined in GeoLIB. The headers of the packets all contain 1) the type of request (i.e., an initialization request (IPC_INIT), a move request (IPC_MOVE), etc.), 2) the data type to which the request refers (i.e., GEO_DT_POINT for points, GEO_DT_POLYGON for polygons, etc.), and 3) the length of the data. In implementing GeoManager, only new IPC_INIT and IPC_MOVE request types were added to GeoLIB, thus requiring only minor additions to the original IPC functions used by GeoSheet to handle the new message types. IPC_INIT is used to inform GeoManager of the requesting GeoSheet's port number, and IPC_MOVE is used by the user program to request that the "mouse move phase" be initiated, which will be explained later.

The need to differentiate between the sources of the packets being sent to GeoManager did not arise, as the request type of the packets helped identify whether the packets being received came from the user program or from GeoSheet. For example, IPC_INIT would never be called by a user program, just as IPC_MOVE or IPC_READ would not be called by GeoSheet. There is an implicit division between the GeoSheet requests and user program requests.

The data flow for the minimal GeoManager configuration (one GeoSheet running, and no "stray objects" placed on it) only flows in one direction:

1. The user program sends GeoManager a packet requesting a certain data object (i.e., `Graphic_Read()`, `Graphic_Write()`, ...).
2. GeoManager acknowledges this packet.
3. GeoManager sends GeoSheet a copy of the same packet.
4. GeoSheet acknowledges this packet.
5. GeoSheet sends GeoManager the data of the requested object type.
6. GeoManager acknowledges this packet.
7. GeoManager sends the user program a copy of this packet.
8. The user program acknowledges this packet.
9. GeoManager saves a copy of this object.

Compared to the original configuration where packets were sent and acknowledged only between the user program and GeoSheet, there are many more packets being sent between the three applications. However, as long as a reliable protocol is used to prevent this increase in packets to overburden GeoManager, even as multiple GeoSheets and multiple user programs become implemented and utilized more, GeoManager should be able to run efficiently.

3.5 Mouse-related functions

To implement an "algorithm move" option in GeoSheet (the "mouse move phase"), a new button was created and added to GeoSheet. If the user wishes to take advantage of this move option, the program must contain an `IPCServiceCleanup()` function at the end of their program (corresponding to the `IPCServiceSetup()` function required

at the beginning of their program to enable the IPC connection to be established with GeoSheet).

The "mouse move phase" consists of the following sequence of events:

1. `IPCServiceCleanup()` sends an IPC_MOVE request, which displays the "algorithm move" button on GeoSheet.

2. When this button is pressed, a series of functions monitors the movements of the mouse, updating the coordinates of the appropriate object in GeoSheet's own object list (an internally-maintained list of all of the objects displayed on GeoSheet).

3. While the object is being moved on GeoSheet, the `Return()` function is called repeatedly (similar to the the `Return` button on GeoSheet which is pressed when the user places an object on GeoSheet and wants to send the data for this object to the user program).

4. The `Return()` function retrieves the data for the updated object in its list to send the appropriate data back to the user program (through GeoManager).

5. The user program updates its GSArguments list with the new data and re-runs the algorithm function.

6. The updated output object is re-sent to GeoManager.

7. GeoManager sends the same message to GeoSheet.

8. GeoManager updates its own copy of the object.

9. GeoSheet deletes the old output and displays the new one

10. The last 7 steps are repeated until the user double-clicks the mouse to end the mouse move and the execution of the user program.

In implementing this "mouse move phase", no changes were made to the original "mouse move" functions. The additional functions for the mouse were based on the original mouse-related functions, but were added to the GeoMAMOS-related code in GeoSheet.

3.6 Multiple GeoSheets

GeoManager handles multiple GeoSheets at once by storing a list of GeoSheet IDs. Whenever a `Graphic_Read()` request is made by the user program, the user-specified GeoSheet in the list sends back an object, and GeoManager then sends a copy of the same object to the user program. When a `Graphic_Write()` request is made for all GeoSheets, GeoManager simply sends a copy of the same request to all of the GeoSheets in the list to display. The only time all of the GeoSheets may not receive a copy of a `Graphic_Write()` request from the user is during a mouse move. A mouse move is allowed by only one GeoSheet, so the user may either specify the GeoSheet that is to display the "algorithm move" button, or let GeoManager send this display request to the default GeoSheet, which is the first GeoSheet listed in GeoManager's list. So when an IPC_MOVE packet is sent, only the selected (or default) GeoSheet in the list will be allowed to move the object, but the output from that updated object may be displayed on all GeoSheets while the mouse is being moved.

3.7 The new GeoMAMOS

GeoManager was developed with only the communication between the user program and GeoSheet in mind. However, GeoManager can be easily incorporated into GeoMAMOS because of its modularity. It is only a background process which provides service at a specified port using GeoIPC, just as all of the other components of GeoMAMOS use GeoIPC. Figure 5 represents the updated GeoMAMOS configuration with GeoManager.

As the figure illustrates, GeoManager has the potential to be used not only between user programs and GeoSheet, but also as a managing interface between all of the GeoMAMOS components. For example, GeoGDB may take advantage of GeoManager by enabling the user to catch bugs in their programs graphically. An instance of this would be while the input objects are being moved on GeoSheet and the algorithm is being debugged. The user could move the input objects around to actually see if there are any "special cases" where the input coordinates cause a miscalculation in the program, thus giving an incorrect output. In this kind of situation, a combination of GeoGDB's breakpoints and GeoManager would help make debugging much easier.

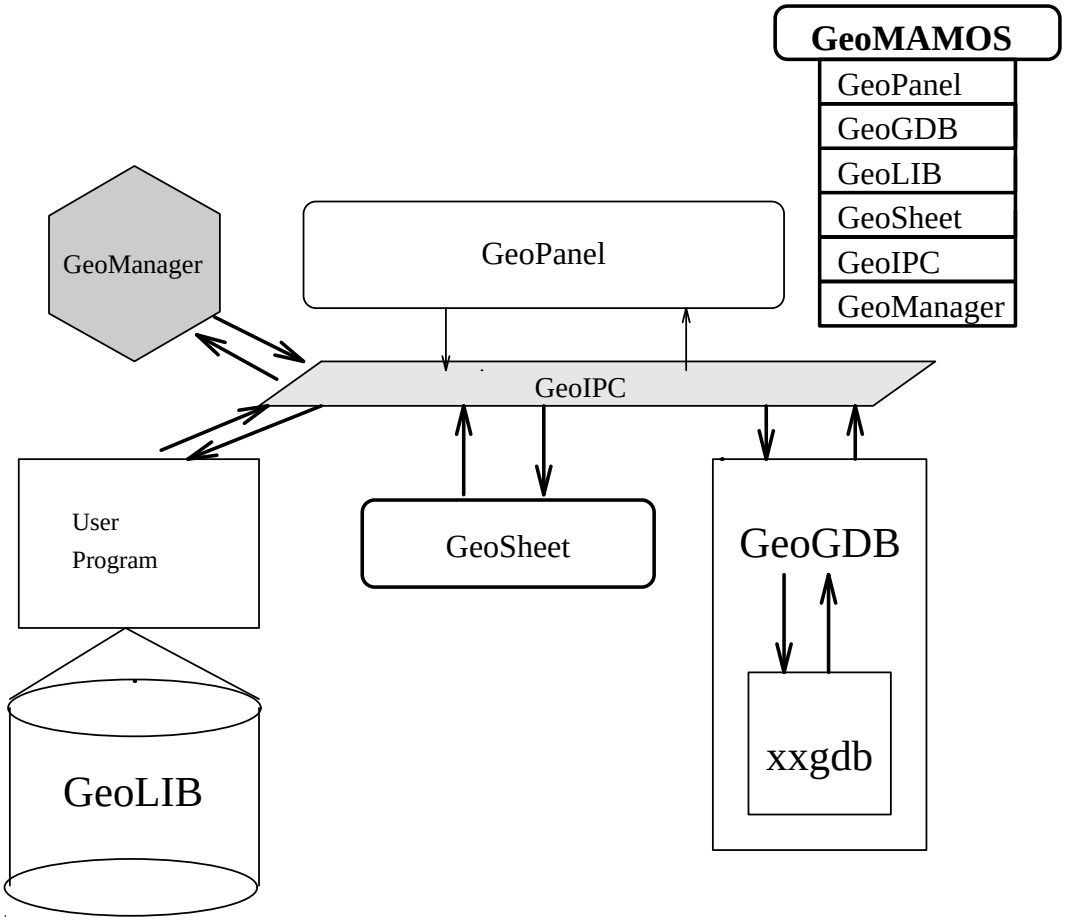

Figure 5: The New GeoMAMOS with GeoManager.

Chapter 4

Sample Execution of a User Program

This chapter will step through the data flow during an execution of a user program which runs the Visibility algorithm using GeoManager.

4.1 The user program

The user program used for this example only contains 37 lines of code. It basically reads a polygon object, reads a point, calculates the visible region within the polygon from the point, outputs the visible region as another polygon, then calls IPCService-Cleanup(), during which the "mouse move phase" is put into effect.

```
 1  #include <geoLEDA/geo_polygon.h>
 2  #include <geoLEDA/geo_polyalg.h>
 3  #include <geoLEDA/geo_algorithm.h>
 4
 5  main()
 6  {
 7    geopolygon *P, *final;
 8    GeoPoint pt1;
 9    GSArguments *gsarglist[2], *temparg;
10    node v;
11
12    IPCServiceSetup("cad2", 9876);
13
14    P = new geopolygon;
15
```

```
16    int i = 0;
17
18    P->Graphic_Read();
19
20    gsarglist[i] = new GSArguments;
21    (gsarglist[i++])->SetObject(P);
22
23    pt1.Graphic_Read();
24
25    gsarglist[i] = new GSArguments;
26    (gsarglist[i++])->SetObject(pt1);
27
28    gsarglist[i++] = 0;
29
30    temparg = Run_alg(&VISIBILITY, gsarglist);
31    final = temparg->Getgeopolygon();
32
33    final->Graphic_Write();
34
35    IPCServiceCleanup();
36
37  }
```

The actual VISIBILITY function is an overloaded function. One of these functions takes the GSArguments list (below) and parses it to pass the appropriate objects to the "real" function which takes a geopolygon and a GeoPoint. Notice that the objects in the argument list `gsarglist` is ordered according to the order in which the objects are passed to the actual function.

```
GSArguments* VISIBILITY(GSArguments** gsarglist) {

    geopolygon* P = new geopolygon;
    P = gsarglist[0].Getgeopolygon();

    GeoPoint* Pt = new GeoPoint;
    Pt = gsarglist[1].GetGeoPoint();

    geopolygon *Final_poly = new geopolygon;
    Final_poly = VISIBILITY(*P, *Pt);

    GSArguments *GSOutput[1];
```

```
    GSOutput[0] = new GSArguments;
    (GSOutput[0])->SetObject(Final_poly);

    GSOutputList = GSOutput;
    return GSOutput[0];
};
```

`GSOutputList` is a global variable which is a list of GSArguments containing the output to the function pointed to by the global function pointer `algname`.

4.2 GeoManager setup

When a user wants to run their program on GeoSheet using GeoManager, the user must verify that GeoManager is running. This is done by actually running the GeoManager executable. If it is already running, a message will be displayed indicating so.

The setup process for GeoManager is very simple. It merely sets up its socket at port 9876, then waits for packets to be sent to it. GeoSheet then sets up its own socket, asks for the hostname on which GeoManager is running, and sends GeoManager an IPC_INIT message which contains its own port number. GeoSheet then waits for packets to be sent to its port from GeoManager. Other GeoSheets can be started up in the exact same manner. GeoManager will simply append the port numbers for the succeeding GeoSheets to its list of GeoSheet IDs.

The user program can then be run, sending its packets to GeoManager, which in turn sends packets to the GeoSheets in its list. Requests from the user program are acknowledged by GeoManager. When GeoManager sends its requests to GeoSheet, GeoSheet acknowledges those. GeoManager saves copies of packets containing object data before passing them on, so that it can store an "official" list of the objects that the user program "thinks" GeoSheet is displaying and the objects that GeoSheet "thinks" it has.

4.3 The program execution

Following the code given in Section 4.1, the flow of data is as follows.

1. (line 18) The user program requests a polygon object.
2. GeoManager acknowledges this request.

3. GeoManager passes the request over to the selected GeoSheet.
4. The selected GeoSheet acknowledges this request.
5. The user inputs a polygon on the GeoSheet editor and presses the `Return` button, which sends a packet containing the polygon's data to GeoManager.
6. GeoManager acknowledges this packet.
7. GeoManager saves a copy of this object, and sends the packet to the user program. An IPC_WRITE request for this object is then sent to all of the other GeoSheets in its list.
8. The other GeoSheets each acknowledge this request in turn.
9. The other GeoSheets display the object.
10. (line 21) The user program appends the polygon to its argument list.
11. (line 23) The user program requests a point object (`Graphic_Read()`).
12. GeoManager acknowledges this request.
13. GeoManager passes the request over to the selected GeoSheet.
14. The selected GeoSheet acknowledges this request.
15. The user inputs a point on the GeoSheet editor and presses the `Return` button, which sends a packet containing the point's data to GeoManager.
16. GeoManager acknowledges this packet.
17. GeoManager saves a copy of this object, and sends the packet to the user program. A IPC_WRITE requests this object is then sent to all of the other GeoSheets in its list.
18. The other GeoSheets each acknowledge this request in turn.
19. The other GeoSheets display the object.
20. (line 26) The user program appends the point to its argument list.
21. (line 30) The user program runs the algorithm, passing it its argument list.
22. (line 31) The output object from the algorithm is saved in a polygon object.
23. (line 33) The user program requests to display this object (`Graphic_Write()`).

24. GeoManager acknowledges this request.
25. GeoManager passes the request over to all of the GeoSheets in its list, waiting for acknowledgement from all of them in turn.
26. The GeoSheets each acknowledge this request.
27. The GeoSheets display the polygon.
28. (line 35) The user program calls IPCServiceCleanup(), which requests an IPC_MOVE.
29. GeoManager acknowledges this request.
30. GeoManager passes this request over to the selected GeoSheet.
31. The selected GeoSheet acknowledges this request.
32. The selected GeoSheet displays the "algorithm move" button.
33. The user presses the button, selects an object and moves it.
34. During the move, GeoSheet repeatedly sends back updated data object values to GeoManager.
35. After sending back the updated output, the user program sends back another IPC_MOVE request (`sendto()` and `recvfrom()` messages must be in pairs, meaning in order to receive updated data the user program must send requests for it each time).
36. GeoManager acknowledges the packets as they are received.
37. GeoManager sends the waiting user program the packets containing the updated data while passing to GeoSheet the IPC_MOVE requests.
38. These last four steps are repeated until the user double-clicks the mouse.
39. The double-click signal tells GeoSheet to send back an IPC_ACK packet instead of an IPC_MOVE packet.
40. GeoManager acknowledges this packet.
41. GeoManager passes this packet on to the user program.
42. The user program terminates.

This sequence of steps is basically the same for all user programs that run algorithm functions defined in GeoLIB. For programs that consist only of `Graphic_Read()` and `Graphic_Write()` and object-manipulation functions that simulate geometric algorithms, because these programs would not be setting any value for the `algname` function pointer, the "mouse move phase" would not execute anything, and the program would simply terminate.

Chapter 5

Discussion and Future Work

5.1 Discussion

This section will analyze the implementation of the project in accordance with the Project Requirements specified in Chapter 2.

5.1.1 Transparency to the user

The addition of GeoManager only requires that the user 1) create a `GSArguments` list for the input objects to the algorithm function being called in `Run_alg(&alg, geolist)`, and 2) include another statement at the end of their program, `IPCService-Cleanup()`, during which the mouse is used to modify the input data, and the algorithm executes with the updated objects. Although there is more work in preparing the input object list, the mouse functions are still transparent, and the rest of the user program can be written just as if GeoManager never existed. So transparency to the user was maintained with the implementation of GeoManager.

5.1.2 Hard-coded algorithms in GeoSheet

The use of global function pointers allows easy access to the algorithm functions defined in the library, without having to enumerate them. Simply calling the function `Run_alg(&alg, geolist)` will execute the specified algorithm function with the corresponding parameter objects. So the modularity of the components of GeoMAMOS is maintained even with the addition of GeoManager.

5.1.3 System independence

No changes were made to the low-level mouse-related code of the graphic editor on which GeoSheet is based. All of the mouse-dependent functions added were only additions to the part of the code relevant to GeoMAMOS. Thus the GeoMAMOS code remains portable to other graphic editors and other operating systems even with GeoManager.

5.2 Future Work

This section will discuss limitations that still exist in GeoManager, and ideas for future projects that can be implemented based on GeoManager.

5.2.1 Concerns

Although GeoManager provides useful functionality to GeoMAMOS, there is still the increase in the number of packets being transmitted between the user program, GeoManager, and GeoSheet. As GeoManager continues to develop, it is my hope that this increase will become less of a factor in analyzing the performance of GeoMAMOS. That is, by weighing the increased functionality of GeoMAMOS overall against this extra overhead, the number of packets should not be of much consequence.

If some way could be found for GeoManager to run the algorithm function as opposed to having the user program run it, then the number of packet transmissions should be decreased considerably, since the actual execution of the algorithm would take place in GeoManager, and the packets being passed between the user program and GeoManager currently during the mouse move would no longer be needed. Instead, the algorithm function can be executed in GeoManager with the updated objects it receives from GeoSheet, and the output of the algorithm can immediately be sent back to GeoSheet. This option could be easily exploited using the `Run_alg(&alg, geolist)` function, as will be explained later.

As the number of packets increases, however, there will be a greater need for error checking. GeoManager is a very good candidate for this job. It is aware of the data types of the objects that the user program is requesting, and it is keeping track of the current objects on GeoSheet. If any "impossible" data objects (i.e., GeoPoints with negative coordinates) are passed from the user program to GeoSheet, GeoManager could detect it before it is passed to GeoSheet, thus preventing GeoSheet from ever seeing it. Although error detection would not be 100%, all of those that may cause

problems in GeoSheet should be detected by GeoManager.

Another concern is in detecting when a GeoSheet or user program goes down unexpectedly. Either a timing mechanism or an "Are you there?" protocol should be adopted in case of such an occurrence.

5.2.2 Possible extensions

Objects which consist of combinations of lower-level objects, such as weighted and unweighted, directed and undirected graphs, should be taken into consideration during the "mouse move phase." Since the components of these objects are normally "attached" to one another (i.e., the nodes of graphs with respect to their edges), when the mouse moves one of these objects, those objects that are connected to it must also change their configurations to the appropriate position. Currently, GeoSheet does not have the capability of detecting such a move. However, GeoManager could be used to direct GeoSheet that such a configuration exists by analyzing the data being passed between the user program and GeoSheet and sending GeoSheet messages of when it should move the positions of multiple objects.

This may cause a delay in GeoManager, however, since it would have to analyze every object in its list every time a move is initiated. Therefore, an efficient protocol would have to be determined in order for GeoManager to maintain its currently prompt responses.

The end of Chapter 3 also mentioned the capability of GeoManager to be used by all of the components of GeoMAMOS. The example given was debugging in GeoGDB. Similarly, GeoManager could be used by GeoPanel, or even by GeoSheet's 3D counterpart, 3D-GeoSheet. Using 3D-GeoSheet, GeoGDB could use GeoManager to analyze 3D objects as a 3D algorithm is being executed. When the program stops at a breakpoint, the user could use the mouse to both manipulate the object and/or move the user's viewpoint by rotating the coordinate axes to a different position.

Another idea is based on the fact that multiple GeoSheets can be handled by GeoManager. That is, currently, there is a GeoSheet:GeoManager:User program ratio of N:1:1. However, there is no way for multiple user programs to be displayed on multiple GeoSheets simultaneously (N:1:M). For example, suppose there are two existing programs that perform certain calculations on weighted graphs. If a user wanted to take the output of one of the programs to use as input to a second program, whose output in turn would be used for the first program, GeoManager would only end up getting confused. In order to implement this, GeoManager could keep track of the

various user programs sending packets to the GeoSheet(s). The data in the succeeding packets may have to take the "user program ID" into consideration, thus requiring the packet format to change, but this idea is not impossible. It may be even easier to implement if GeoManager could be made to store the global `algname` function pointer, which brings us back to the earlier discussion of sending GeoManager the pointer to the algorithm function, perhaps through `Run_alg(&alg, geolist)`.

If `algname` could be used by GeoManager, then both the GeoManager project and other project implementations would be simplified. However, because `algname` is a pointer, it cannot be passed from the user program to GeoManager through IPC as is. Enumerating all of the algorithm functions in GeoLIB in a list could make this work. A list of "algorithm function IDs" which assign each algorithm function a unique ID could be maintained (by whom is another concern). Then the algorithm function ID corresponding to the ID `algname` would be passed in a packet from the user program (while in `Run_alg(&alg, geolist)`) to GeoManager. GeoManager can simply look up this function, set its own `algname` variable to point to it, then run it with its own list of input objects that it has been maintaining. However, this idea requires that the user edit the algorithm function ID list every time a new algorithm function is added to the library (unless this process can be automated). The best solution would be to avoid having this algorithm function ID list, so the real challenge is to inform GeoManager of the function pointed to by `algname` *without* enumerating all of the algorithm functions available in GeoLIB.

5.2.3 Other ideas

Because the "mouse move phase" continuously re-executes the algorithm *while* the mouse is moving one object, GeoManager may not be useful for moving multiple objects at once. For example, a user may wish to move a line segment as well as a point on the GeoSheet screen, but the moving of one before the other may cause a problem in the algorithm if it is executed in the middle of this move. Therefore, another button which can be displayed with the IPC_MOVE request could be a "Wait Move" button, which, when pressed, can allow the user to first move the appropriate objects on GeoSheet, then press the `Return` button to re-execute the algorithm. This could be repeated anytime during the "Wait Move Phase" until the user double-clicks the mouse, which re-executes the algorithm one last time before terminating the program. This idea may require some kind of labeling system for the objects, so that the user program would know which object(s) were moved to update the appropriate object(s) when it passes the object list to the algorithm function.

Another idea is to have some sort of recording mechanism, which records the algorithm functions that were called during some set period of time. So if a user is developing a new algorithm to solve a certain problem, the order of the algorithms that the user experiments with could be recorded. This would be useful as a tool to keep track of what algorithms were tested as well as to have a record of the order of algorithms that produced a certain output.

Finally, another button could be created which would allow different GeoSheets to be able to move the mouse at one time. This would make the multiple GeoSheets more interactive. GeoManager would keep track of a "mouse sempahore" which allows only one GeoSheet to be the "Master" GeoSheet during the "mouse move phase." A button could be pressed when one of the "slave" GeoSheet users wishes to move an object on GeoSheet. However the appropriate Move button would not be available to this user unless no other GeoSheets are using the mouse. This idea, if implemented, would be very useful especially for those communicating at remote sites and wish to illustrate something visually. The current `GeoPause()` function, which displays messages on the screen, can also be very useful in this environment.

Chapter 6

Conclusion

The GEOmetric MAnipulation and MOnitoring System, up until now, was not able to provide much MAnipulation of geometric algorithms per se. However, the GeoManager Project was able to finally provide that functionality, and it should prove to be very useful from both educational and research perspectives. A teacher would be able to demonstrate various algorithms simultaneously to students sitting at different terminals, while a researcher developing algorithms would be able to experiment with various input data to their algorithms much more efficiently than before. Many other uses for GeoMAMOS with GeoManager are possible.

The GeoManager Project should not end here, however. The potential of GeoManager can be taken advantage of with little effort, as was described in Chapter 5. I hope to see GeoManager grow to allow users to take advantage of the comprehensiveness of GeoMAMOS even more easily.

Bibliography

[1] Richard M. Adler, "Distributed Coordination Models for Client/Server Computing," *Computer*, April 1995, pp. 14-22.

[2] V. Anupam, C. Bajaj, D. Schikore and M. Schikore, " "Distributed and Collaborative Visualization"", *Computer*, July 1994, pp. 37-43.

[3] M. Accetta, et al., "SHASTRA: A Distributed and Collaborative Design Environment", *Animation of Geometric Algorithms: A Video Review*, June 1992, pp. 12-14.

[4] P. J. de Rezende and W. R. Jacometti, "GeoLab: An Environment for Development of Algorithms in Computational Geometry". *Proc. 1st Int'l Computational Geometry Software Workshop*, Geometry Center, MN, Jan. 18-20, 1995; see also *Proc. 9th Symp. on Computational Geometry*, May 1993, pp. 401-402.

[5] Epstein, P., J. Kavanagh, A. Knight, J. May, T. Nguyen, and J.-R. Sack, "A Workbench for Computational Geometry, " *Algorithmica,* April 1994, pp. 404-428.

[6] D. T. Lee, "Visibility of a Simple Polygon," *Computer Vision, Graphics, and Image Processing,* 1983, pp. 207-221.

[7] D. T. Lee and A. K. Lin, "Computational Complexity of Art Gallery Problems," *IEEE Trans. Infor. Theory,* IT-32,2, March 1986, pp. 276-282.

[8] D. T. Lee, S. M. Sheu and C. F. Shen, "GeoSheet: A Distributed Visualization Tool for Geometric Algorithms", *Computational Geometry Software Workshop*, 1995.

[9] S. Näher, "LEDA – A Library of Efficient Data Types and Algorithms", Max-Planck-institut für informatik. Technical Report A 04/89, Universität des Saarlandes, Saarbrücken, 1989.

[10] S. Näher, "LEDA3.0 User Manual", technischer Bericht A, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1992.

[11] O'Rourke, J. Art Gallery Theorems and Algorithms, New York, Oxford, Oxford University Press, 1987.

[12] The Xfig User Manual. 1993.

[13] P. Schorn, "An object oriented workbench for experimental geometric computation," *Proc. 2nd Canadian Conference in Computational Geometry*, Ottawa, August 6-10, 1990, pp. 172-175.