A Simple Concept for Distributed Computing in Computer Graphics

Bernd Oberknapp
Sonderforschungsbereich 256
Institut für Angewandte Mathematik
Universität Bonn
bo@iam.uni-bonn.de

Konrad Polthier
Sonderforschungsbereich 288
FB 3 Mathematik, MA 8-3
Technische Universität Berlin
polthier@math.tu-berlin.de

Abstract

We propose a simple concept for distributed computing. In a first stage this allows stand-alone programs to communicate and exchange data across computer networks. Existing stand-alone programs need very little adaptation to participate in such a system of (temporarily) connected programs. In a second stage the concept is extended to allow remote-objects and remote-methods. All network related functionality is located in the network manager, therefore adapted stand-alone programs may still run stand-alone without the network manager.

CR Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems — *Distributed applications*; I.3.2 [**Computer Graphics**]: Graphics Systems — *Distributed/network graphics*; I.3.4 [**Computer Graphics**]: Graphics Utilities — *Application packages*;

1 Working Together and Distributed Computing

In the mathematical community exist a number of excellent software packages developed at different sites in the world. Each package provides fairly complete functionality for specific mathematical problem areas but surely none of them is universal or has the aim to be.

Our net-manager concept is a general approach to combine the functionality of specialized stand-alone programs to a net of packages. Inside the net different packages may have connections to others, exchange messages and transport arbitrary data through *information channels* (figure 1). The programs may run on different computers with their display directed to different terminals, therefore several people may use the programs at the same time and exchange information with other programs through the information channels.

An important feature of the concept is that programs need very little adaptation for working inside the net and that they still have the ability to run stand-alone even though they are used as modules in the net. This is a major difference to other net concepts as for example the network system in AVS [AVS89] or PVM [PVM94] (see section 5).

The net-manager was created to extend the capabilities of GRAPE [Gra95], [OP94], the object-oriented GRAphics Programming Environment developed at the Sonderforschungsbereich 256 in Bonn (see section 6.1), by adding *remote objects* (figure 2) and *remote methods*. Objects in the data hierarchies of GRAPE programs running simultaneously on different computers, e.g. a c_grape on a mainframe for large computations and a g_grape on a graphic machine for post-processing (this can of course be the same executable running on both machines), can be linked together using the net-manager. The c_grape performs large computations and updates the hierarchy node of the g_grape after each computation cycle or on request from the g_grape. The user at the graphic machine may post-process the data as usual when working with GRAPE, i.e. apply rotation, clipping and different display methods on the data set (figure 3). But he has to be aware that the data is updated from time to time by the computation process on the mainframe, in this sense the object at the hierarchy node of the g_grape is a remote object. The user can control the computation on the mainframe by sending methods in g_grape to the node which represents the remote object, these remote methods are executed by c_grape.

This kind of application which allows the parallelisation and distribution of computations was the initial

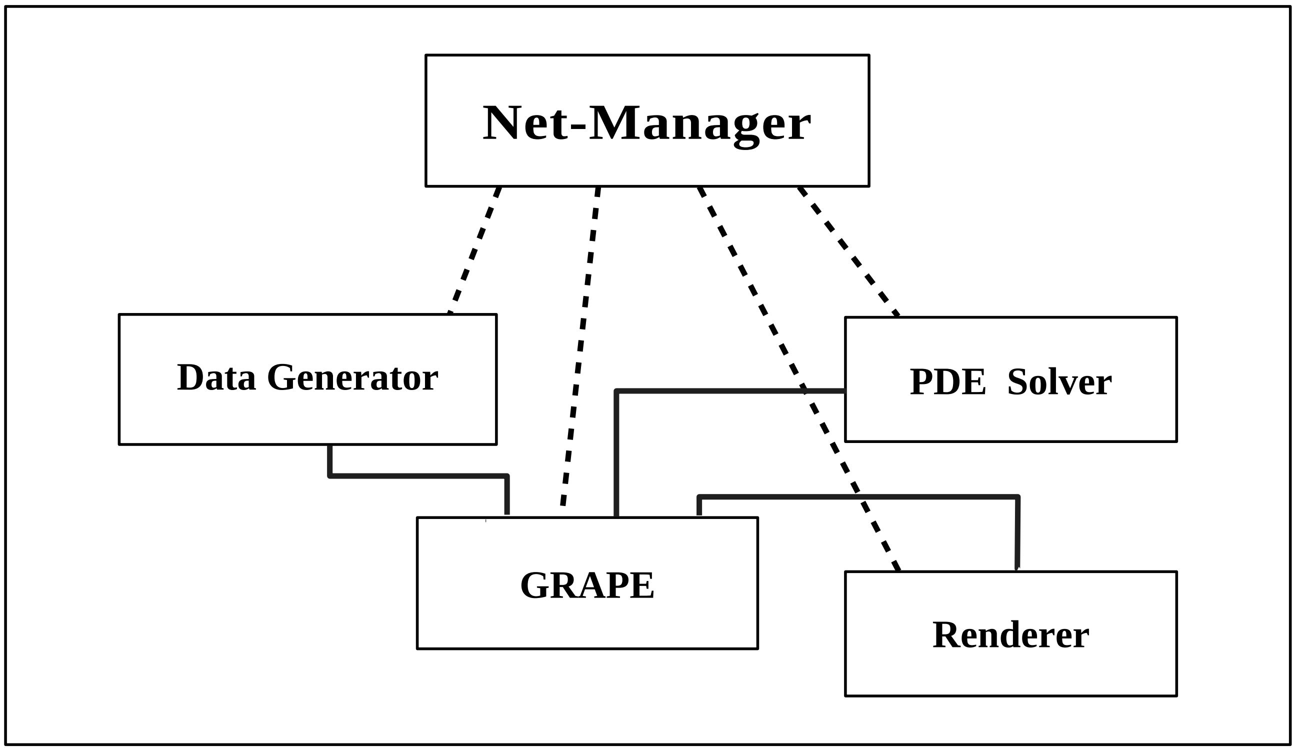


Figure 1: Information Channels. The net-manager creates and supervises connections, it has a command connections to every program running under its control. The programs exchange data directly through information channels, this enables them to exchange arbitrary data. The connections are created by the net-manager temporarily on request. The four programs shown in this figure are just examples for possible stand-alone programs connected via the net-manager.

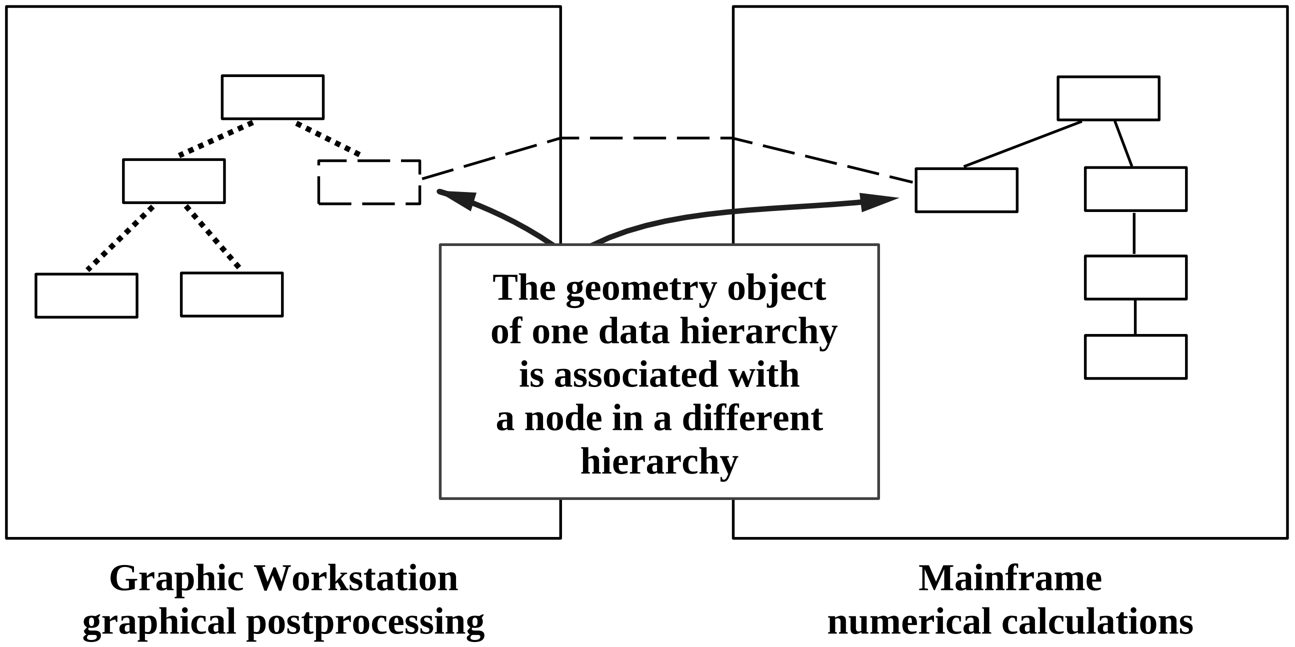

The geometry object of one data hierarchy is associated with a node in a different hierarchy
Graphic Workstation
graphical postprocessing
Mainframe
numerical calculations

Figure 2: Remote Objects. Two separate programs on a graphic machine and a mainframe have the same geometric object in their data hierarchies. In the example the object physically belongs to the mainframe and is a remote object for the graphic machine, but the user on the graphic side sees no difference when accessing the object for display or other purposes.

reason to start working on the net-manager. The application of linking separate main programs to a net was the first stage in this development, it came out as a by-product of the more general aim of allowing remote objects and methods. But from the user's point of view linking separate programs is the first demand he makes on a net-manager: without bigger changes he can work with the programs he is used to at a higher level of interactivity. After having gained experience with this kind of distributed computing and working together of programs the user will much more efficiently apply the concept of remote objects and remote methods in an environment like GRAPE.

Let's now discuss the working principles of the net-manager in more detail. The description is first restricted to the linking of separate programs, remote objects and methods are discussed in the last section.

2 The Net-Manager Concept

The net-manager was designed with these aims in mind:

- Existing programs should need only small adaptations for working inside the net, the net specific parts should be encapsulated from the rest of the program.
- Programs adapted for the net-manager should still be able to run stand-alone without the net-manager in the background.
- Programs should be able to exchange arbitrary data, not only some predefined data types the net-manager knows.
- It should be possible to run programs stand-alone first and then start the net-manager or connect to an already running net-manager.
- The concept should be extendable to hierarchies of nets with local net-managers on different machines.
- The net-manager should be easily configurable for the user.

To realize these aims we developed the following concept for the net-manager:

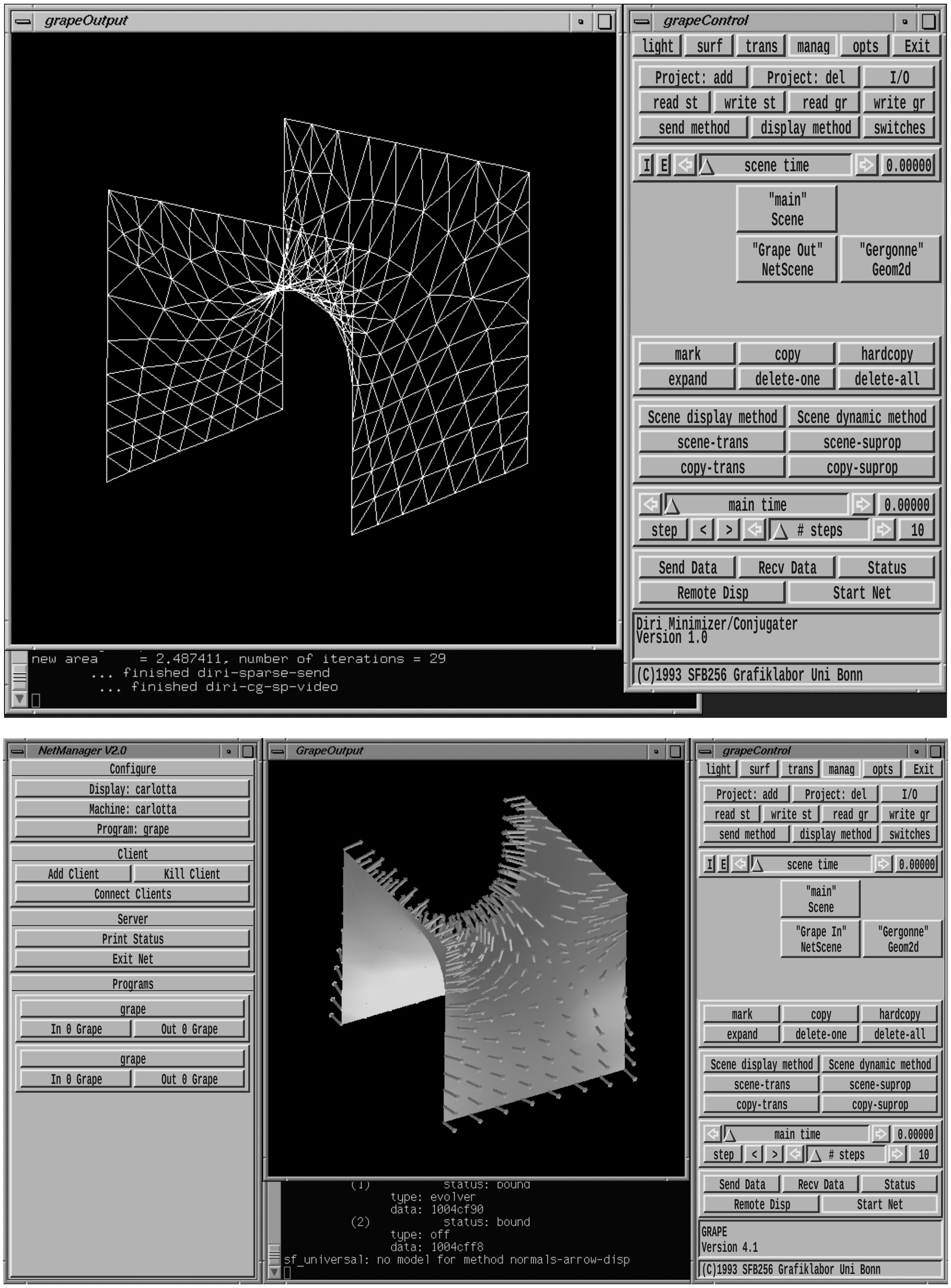
- The net-manager is the control instance for building and maintaining the net of programs. Programs can be added to and removed from the net by starting them or telling them to quit, the machines they run on and the displays they use can be configured. Information about the available programs, machines and displays is provided by the user.
- Programs running under the control of the net-manager communicate with the net-manager over a permanent command connection which is created when the program is started by the net-manager or when a running program connects itself to the net for the first time and deleted when the program is disconnected from the net.
- To send or receive information programs can create input and output ports which are registered with the net-manager. Each port has a fixed type which determines what kind of data it is able to handle.
- The net-manager coordinates and supervises all data exchange in the net. The net-manager creates *information channels* between programs by connecting their input and output ports. The port types are used to determine which connections are valid. The programs are informed that their ports are connected, but the knowledge about the net of connections is kept within the net-manager.

The programs only have information about the net-manager and their own ports, they don't know anything about the other programs and the connections in the net, not even where their own ports are connected to.

If a program wants to write some data to one of its output ports it has to ask the net-manager where the data should be send. The net-manager will check if the program the port is connected to is able to receive the data and tell it to listen to the corresponding input port. The address of this port is returned to the sending program which then is able to write the data. The net-manager is notified after the data transfer has been completed. During the transfer the net-manager is free to handle other requests.

The way the data transfer between programs is handled is essential to make the adaptation of new programs to the net-manager as simple as possible: programs don't communicate with each other directly but only with the net-manager, therefore they don't have to supervise their connections or to manage information about the programs they are connected to. They just set up the control connection with the net-manager and create their ports, the management of the information channels is done by the net-manager.

On the other hand the data is transferred directly between the programs — despite the fact that they know almost nothing about their connections. Therefore the net-manager doesn't need to know which data format is used by the programs. This makes it possible to exchange arbitrary data. The net-manager only



```
new area     = 2.487411, number of iterations = 29
        ... finished diri-sparse-send
         ... finished diri-cg-sp-video
```

```
        (1)         status: bound
                type: evolver
                data: 1004cf90
        (2)         status: bound
                type: off
                data: 1004cff8
sf_universal: no model for method normals-arrow-disp
```

Figure 3: Remote Objects in GRAPE. The computation process (top) is running on an SGI Challenge with X11 control display, the graphical postprocessing (bottom, see color plates for color image) is done on an SGI Indigo2. The same geometry is accessed in both data hierarchies.

needs to know that the ports of the programs are able to handle the same data type.

Another advantage is that the data flow makes no detour over the net-manager. This is important if two programs run on one machine and the net-manager is located on a different machine. In this case the data can be transferred through a local connection or even in memory.

Because programs only communicate with the net-manager it is easily possible to extend the concept to hierarchies of nets, for example with local net-managers on different machines. In this case the programs don't have to be changed at all, only the communication between the net-managers has to be added and the management of the information channels within the net-managers has to be changed.

There are several ways the net-manager can be used: it may be started first and then build up a net of programs, or a single application may be started and later, if necessary, the net-manager may be invoked to connect the running program with others. The later possibility is usually chosen when unforeseen situations occur in the currently running program, making it necessary to perform actions on the data which are supported only by other programs. In both cases the command connection between the net-manager and the application can be build without problems: the one that is started first provides the other one with the necessary information. This is more difficult if the net-manager and the application are already running, in this case the user has to tell the application where it can find the net-manager to enable it to build up the command connection.

Next we will take a look at our current implementation of the net-manager concept before the adaptation of programs is explained.

3 Implementation of the Concept

The most important aspect of the implementation was to design an interface between the network code and the application which encapsulates all net specific parts. This helps to keep the programs free from (probably machine dependent) direct network functions call and makes the adaptation to the net-manager much easier.

The interface code has to be linked to the application. Provided are functions for

- starting the net-manager if the program is running stand-alone,
- creating and deleting the command connection to the net-manager,
- creating, deleting and handling input and output ports and
- sending and receiving data.

If for example a program wants to write some data to one of its ports it only has to call the function for sending data with the port information as parameter. This function handles the communication with the net-manager, sets up the connection with the other program and sends the data. The return value of the function indicates if the transfer was successful.

The messages between the net-manager and the programs are exchanged via sockets, as is the data between programs. This is the only way information can be exchanged between programs running on arbitrary machines in the net. If programs run on the same machine pipes or shared memory can be used.

For transferring messages between the net-manager and the programs in a machine independent format we use XDR [XDR], the external data representation which is part of the RPC package developed by Sun Microsystems. This format should also be used for the data exchange between programs if they are running on different hardware platforms.

An important point is how requests from the net-manager are handled within the program. If the program had to monitor its command connection for incoming messages it would be necessary to change some important functions (e.g. the main event loop), during long computations the program wouldn't be able to react to requests from the net-manager at all. Therefore we decided to use a signal handler which is invoked automatically by the operating system whenever a message from the net-manager arrives.

This is what happens when the program receives a request: the operating system will interrupt the running program, save its current state and execute the signal handler. Now the message can be evaluated, if the program is for example told that a connection to one of its ports was created it will store this information and send a reply to the net-manager. Then the control is returned to the operating system which resumes the execution of the program where it was interrupted.

Like the interface code the signal handler is linked to the program, it is installed when the command connection to the net-manager is created.

The net-manager itself was implemented with the aim to keep the network code encapsulated from the user interface (currently we are using a simple graphical interface implemented with the GRAPE library, see figure 4). This was realized by using the same mechanism like the programs to handle incoming re-

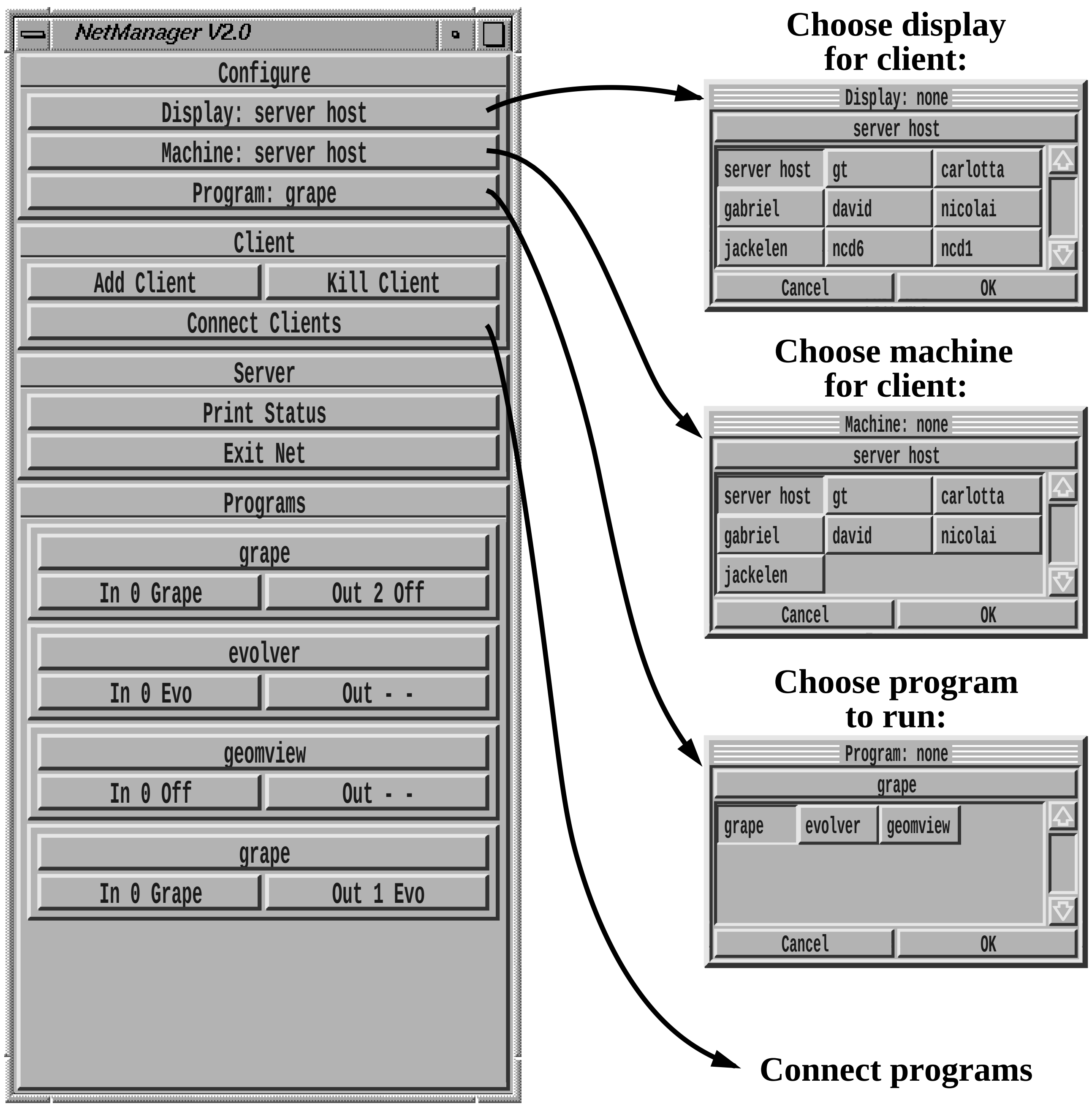


Figure 4: Net-manager interface created with the GRAPE library. Other interface builders could also be used.

quests, a signal handler, and creating an interface to the network code which contains functions for

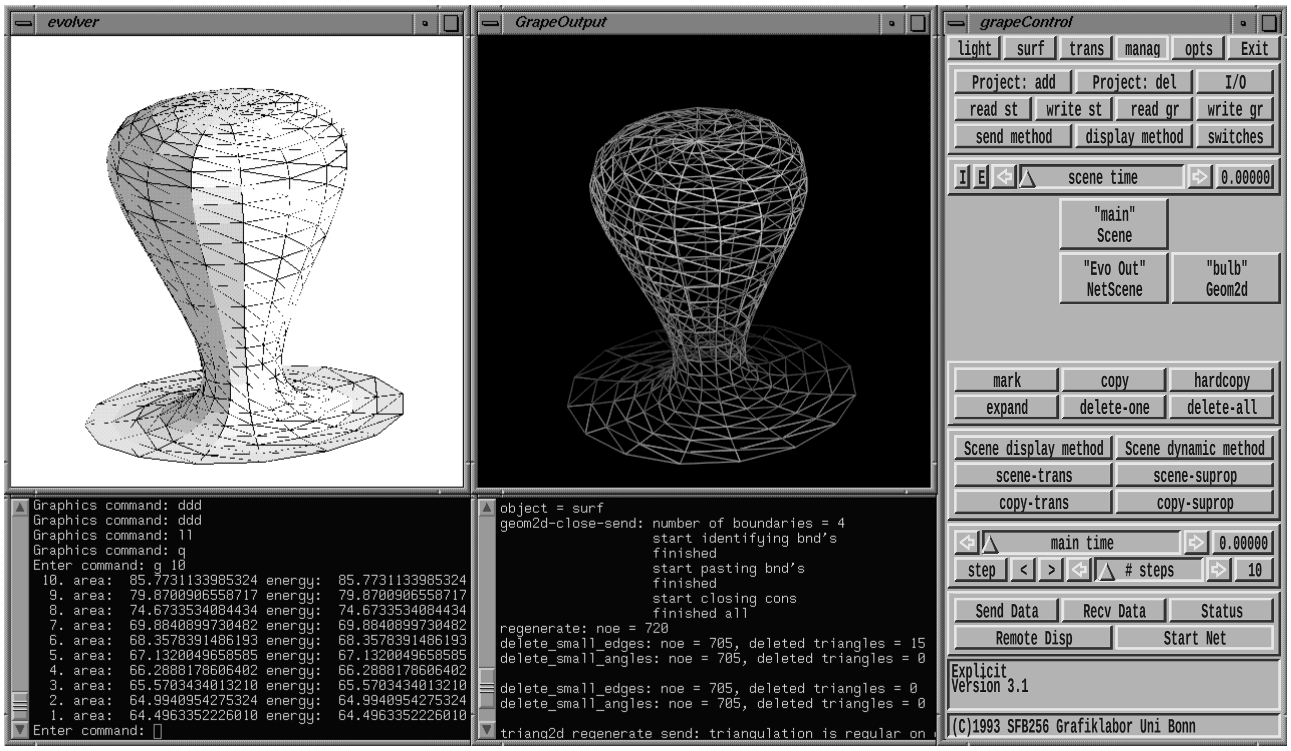
- starting the net-manager,
- managing programs, machines and displays,
- adding programs to and removing them from the net and
- connecting or disconnecting program ports.

Information about the available programs, machines and displays has to be provided by the user in a configuration file. The net-manager is started by reading the configuration file and installing the signal handler.

4 Adapting Programs to Run within the Net

As mentioned before only small changes have to be made to an existing stand-alone program to adapt it for running under the control of the net-manager: only the function calls for creating the command connection to the net-manager and for installing ports (one line of code each) have to be added. The programm still will be able to run without the net-manager, in this case the added function calls just return without doing anything.

Surely data conversion and transfer routines require



```
Graphics command: ddd
Graphics command: ddd
Graphics command: ll
Graphics command: q
Enter command: g 10
 10. area:  85.7731133985324 energy:  85.7731133985324
  9. area:  79.8700906558717 energy:  79.8700906558717
  8. area:  74.6733534084434 energy:  74.6733534084434
  7. area:  69.8840899730482 energy:  69.8840899730482
  6. area:  68.3578391486193 energy:  68.3578391486193
  5. area:  67.1320049658585 energy:  67.1320049658585
  4. area:  66.2888178606402 energy:  66.2888178606402
  3. area:  65.5703434013210 energy:  65.5703434013210
  2. area:  64.9940954275324 energy:  64.9940954275324
  1. area:  64.4963352226010 energy:  64.4963352226010
Enter command: 
```

```
object = surf
geom2d-close-send: number of boundaries = 4
                   start identifying bnd's
                   finished
                   start pasting bnd's
                   finished
                   start closing cons
                   finished all
regenerate: noe = 720
delete_small_edges: noe = 705, deleted triangles = 15
delete_small_angles: noe = 705, deleted triangles = 0

delete_small_edges: noe = 705, deleted triangles = 0
delete_small_angles: noe = 705, deleted triangles = 0

triang2d regenerate send: triangulation is regular on
```

Figure 5: Surface Evolver [Bra92] and GRAPE. To adapt the evolver for the net-manager only 9 lines of code had to be added to the evolver's main program. This modified evolver version still runs stand-alone as before.

the most work when adapting a new program. We have created functions to handle several geometric data formats, for GRAPE objects a rather complete set of transfer functions exists. They all use XDR [XDR] to ensure that the data can be exchanged between different hardware platforms without problems.

If a program is able to handle one of the supported data formats only the command connection with the net-manager has to be set up (to be able to send and receive requests), and ports for exchanging data have to be created. This can be done by adding a few lines of code to the main program as for example:

```
/* private data structure */
struct geometry out;
/* port identifier */
int out_id;

/* create command connection */
net_connect_to_manager();
out_id = net_create_outport(
            &out, DT_GEOMETRY);
```

The function `net_connect_to_manager` creates the command connection to the net-manager and installs the signal handler — this has to be done before any other net-manager function is called. `net_create_outport` creates an output port and links it with the address of the geometry structure, the net-manager is informed that an output port of type `DT_GEOMETRY` was created.

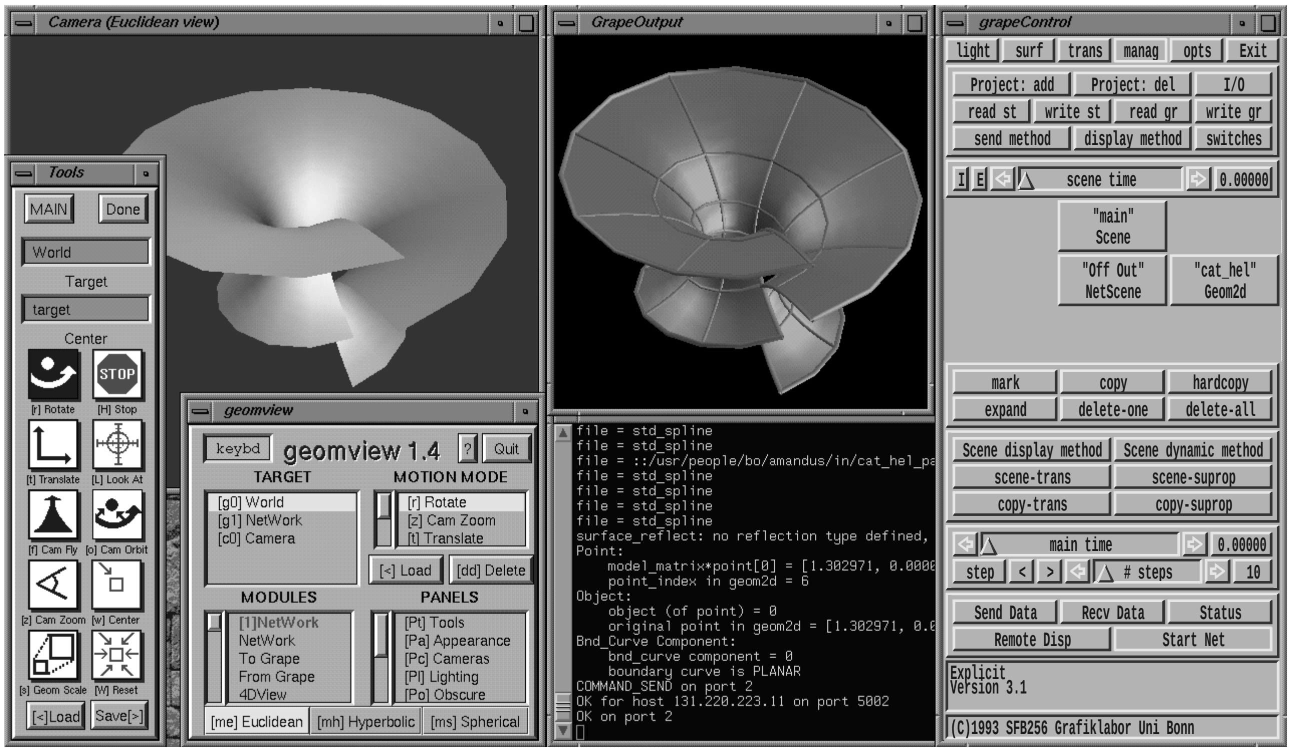
Now the port can be connected by the net-manager to some input port. The type identifier is used by the net-manager to determine which connections are valid, it is also used by the program interface code to decide what kind of data has to be send:

```
net_send_data(&out);
```

Each data structure can be linked to exactly one input port and any number of output ports. `net_send_data` will write the structure to all connected output ports, to select a single port for writing the port id returned by `net_create_outport` can be used.

The user isn't restricted to using the predefined datatypes. One possibility to transfer other types is to convert them to one of the supported types and then to use the existing transfer functions. Of course this conversion has to be done every time data is send or received.

A more elegant but also more difficult way is to implement new data transfer functions. Functions like `net_send_data` or `net_receive_data` which manage the data transfer call functions for writing data to or reading it from an XDR stream. For example the write function could be replaced to handle the private datatype:



```
file = std_spline
file = std_spline
file = ::/usr/people/bo/amandus/in/cat_hel_pa
file = std_spline
file = std_spline
file = std_spline
file = std_spline
surface_reflect: no reflection type defined,
Point:
    model_matrix*point[0] = [1.302971, 0.0000
    point_index in geom2d = 6
Object:
    object (of point) = 0
    original point in geom2d = [1.302971, 0.0
Bnd_Curve Component:
    bnd_curve component = 0
    boundary curve is PLANAR
COMMAND_SEND on port 2
OK for host 131.220.223.11 on port 5002
OK on port 2
```

Figure 6: Geomview [Geo95] and GRAPE (see color plates for color image). An external Geomview module was used for connecting Geomview to the net-manager.

```
int net_write_data(XDR *xdrp,
                   PORT *port)
{
  xdrp→x_op = XDR_ENCODE;
  if(port→type == DT_PRIVATE) {
    xdr_private(xdrp, port→data);
    xdrrec_endofrecord(xdrp, 1);
    return 1;
  }
  return 0;
}
```

Of course the read function of the receiving program also has to be replaced, additionally the `DT_PRIVATE` identifier must be declared to the net-manager in the configuration file. For complex data structures the most difficult part is writing the xdr function (`xdr_private`) which converts the data between its C representation and the network format.

Before presenting the extensions we have added to GRAPE by using the net-manager, we will discuss the differences between our net-manager concept and other network systems.

5 Differences to Other Net Concepts

There are some major differences between our net-manager concept and dataflow packages like AVS [AVS89] or Explorer. These systems use a network manager to interactively combine modules to a program. Data is modified at each module node and then passed over connections to other nodes. The modules have to be designed for working inside the network, they are not able to run stand-alone. The datatypes they can exchange are limited to types known by the system, for AVS for example there is only one (very complex) datatype. The conversion of an existing stand-alone program to a module for these systems will be impossible in many cases, the internal data structures would have to be changed or conversion routines to be added, and the ability to run stand-alone would be lost. With our concept the (stand-alone) programs are running simultaneously under the control of the net-manager, they can exchange any type of data they are able to handle. Only small changes have to be made to adapt programs for running inside the net, in many cases this should be easily possible.

A widely used network package is PVM (Parallel Virtual Machine) [PVM94]. It allows a heterogeneous network of parallel or serial computers to appear as a single concurrent computational resource. PVM was designed to allow hundreds of computers and thousands of task to solve problems in parallel. It contains routines for configuring the virtual machine and for communication between processes. As you might have noticed the functionality provided by our net-manager is a (small) subset of the functionality available with PVM, but the interface to the network for both server and clients is quite different.

Since we were not interested in very large configurations but in working interactively with several programs running on different machines we used a simple server/client model. In fact we could have used PVM for starting clients and for the internal communication between server and clients, but this would have saved us only from writing a small part of the net-manager code.

The authors of PVM avoided to use signal driven I/O and interrupt handlers, mainly because of portability problems. PVM programs have to check regularly if there are messages for them — this is exactly what we wanted to avoid, because important parts of existing programs would have to be changed, for example the main event loop and computation functions (otherwise the program would not react to requests during long computations). Therefore we used the signal mechanism, so far our net-manager has been tested on SGI workstations, IBM risc machines, a Convex supercomputer and Linux PC's. The portability problems mainly seem to occur for machines not running Unix, especially for parallel machines with multiple processors.

The way the data transfer is handled in PVM is a problem for our kind of application. For each datatype pack and unpack routines are needed to encode the data before sending it and to decode it after the transmission. For simple messages like the ones exchanged between server and clients these routines are easy to write, but for very complex geometric data structures (there are dozens of geometric datatypes and classes in GRAPE) this get difficult and very time-consuming. Integrating existing read/write routines in the PVM message system is not possible.

As explained before we use XDR [XDR] not only to transfer messages between server and clients but also to exchange data between clients (the pack and unpack routines of PVM internally use XDR, too). These XDR routines can also be used to archive data in a machine independent format, and they can be used when the program is running stand-alone, too. For example we have written an XDR archiving system for GRAPE which can be used both to store data on disk, tape etc. and to exchange data between GRAPE programs with the net-manager (this GRAPE XDR format is one of the predefined data formats). With our net-manager concept programs can use existing archiving routines to transmit data — the only requirement is that these routines can write the data to resp. read it from a stream. This helps to keep the adaptation of programs to the net-manager as simple as possible.

Finally, to show how the net-manager interface can be integrated in an application, we present the extensions we have added to GRAPE by using the net-manager.

6 Special Net-Features in GRAPE

We have created a variety of tools to handle *remote objects* and *remote methods* in GRAPE and to invoke other programs to act on objects in GRAPE. The interface to the net-manager is the same like for other stand-alone programs but the internal handling of ports and message sending is incorporated into the object oriented design of GRAPE and its hierarchical data organization.

First let us briefly introduce the object-oriented graphics proramming environment GRAPE, a package both authors have been working on for several years.

6.1 Introduction to GRAPE

GRAPE is a GRAphics Programming Environment to understand, explore and solve mathematical problems from differential geometry and continuum mechanics, two of the main research areas of the Sonderforschungsbereich 256 (SFB) for Nonlinear Partial Differential Equations at the University of Bonn. It has been developed by students and staff members of the SFB in Bonn since 1987, but now also receives much impetus from groups in Freiburg and Berlin.

GRAPE has an object-oriented kernel written in standard C and uses a device independent approach for interactive graphic visualization. Therefore it runs on a variety of computers — drivers for many systems are available, including Silicon Graphics, Sun, standard X-window, PostScript, Softimage and others. The system offers a class and method library for mathematical applications and objects for interactive control to be used in C programs. Additionally, it includes an interactive visualization and programming environment. GRAPE is an open system with easy extension of existing classes, inclusion of new methods, and adaptation of the graphics interface. GRAPE allows nearly all geometric objects including their mathematical data to be time-dependent.

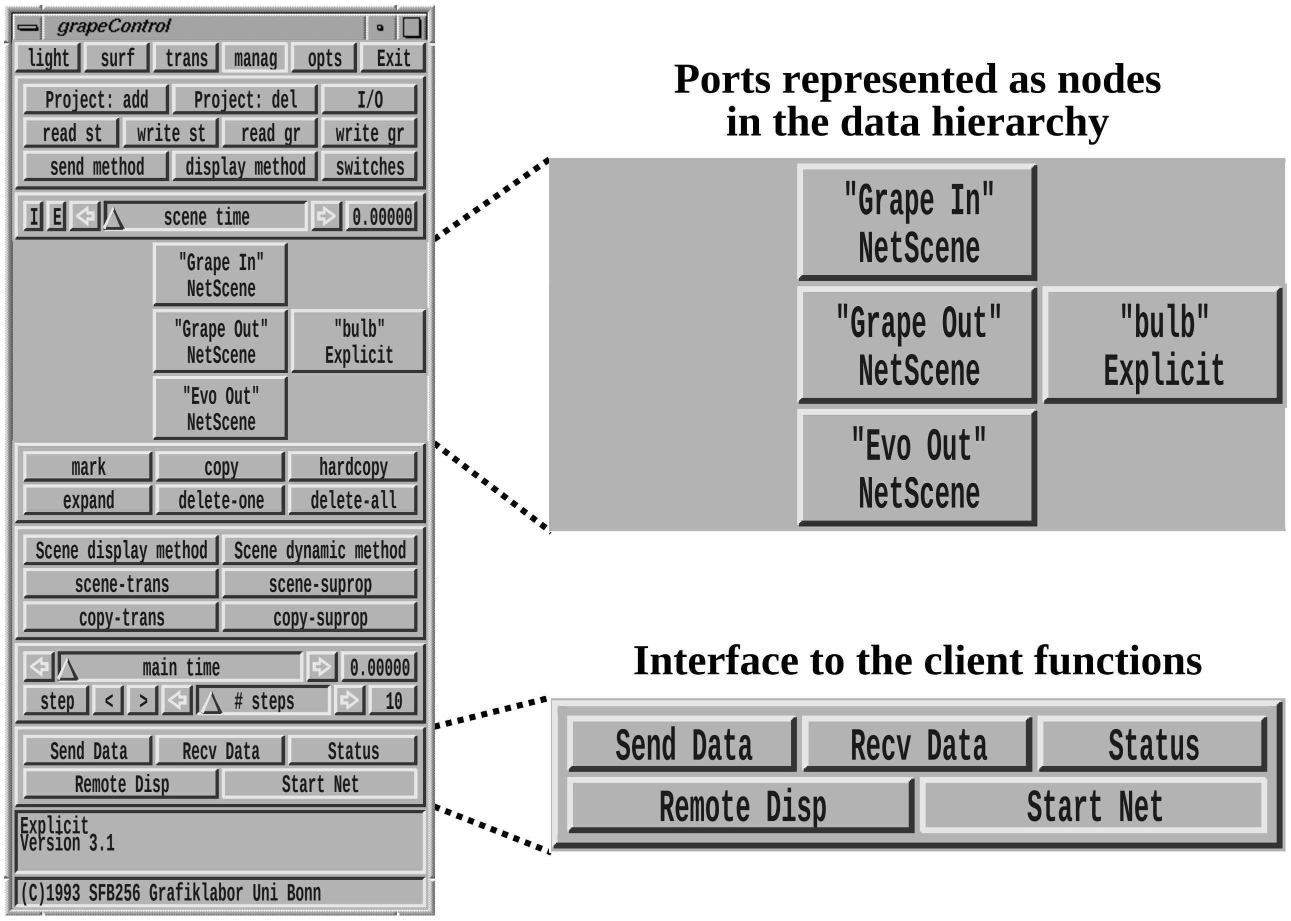


Figure 7: Interface in GRAPE to the net-manager functions. The net-manager is already running, therefore the start button is disabled.

Supported are among other things different kinds of curves, surfaces and volumes in three space and hierarchies of data. Geometric objects may carry additional information as for example an n-dimensional finite element function or describe maps between surfaces. Various mathematical algorithms from differential geometry and finite element theory as well as numerous visualization tools are available including for example local refinement, extraction of level sets, reflection operations, calculation of particle traces or deformations of surfaces in flow fields. The driver concept allows display and modification of objects e.g. in hyperbolic space.

The GRAPE software is non-commercial. Scientific sites may obtain it on request from the SFB in Bonn for free. For more information see http://www.iam.uni-bonn.de or send an email to grape@iam.uni-bonn.de.

6.2 Remote Objects and Methods

In GRAPE each port is represented as a node of a special class `NetScene` in the ordinary data hierarchy (figure 7). The object at this node is in its nature a proxy object. It resides on the remote machine in the remote process. All methods sent to this object are passed to the remote object. This mechanism runs in the background and not visible for the user — for the user the object behaves like a local object. If the remote object of a `NetScene` shall be visualized in the local process we ask the remote object for a copy which is then stored at the local node (figure 3). Visualization methods send to the `NetScene` then operate on the local object for efficiency reasons.

All functions for creating or deleting ports and sending or receiving data were embedded in methods, therefore they can be called by just pressing a button. This includes starting the net-manager if the GRAPE process is already running stand-alone (figure 7).

The principles of remote objects and methods make it possible to execute methods on arbitrary computers chosen at run-time. This allows the user to parallelize his work by doing big computations on other machines and keeping the resources of his computer free for the interactive work:

Assume that you have a large computation job which you want to put on a supercomputer without a graphic display. To control the state of your numerics you can start GRAPE on a graphic computer, connect both programs using the net-manager and let the numerics always send in-between results to the graphic computer. In GRAPE the results will appear as objects at the hierarchy nodes connected to the numerical program and you may operate on the data as you would do in a stand-alone program (figure 3). The only difference is that from time to time the numerics send new geometries updating the nodes' objects.

The graphic computer may even be a simple PC not capable of holding the whole numerical process in its own memory but capable of holding some extracted reduced set of information, for example a section of a dynamic process.

References

[AVS89] C. Upson et al: *The Application Visualization System: A Computational Environment for Scientific Visualization,* IEEE Computer Graphics and Appl., July 1989

[Bra92] K. Brakke: The surface evolver, *Exp. Math. 2,* 141-165 (1992).

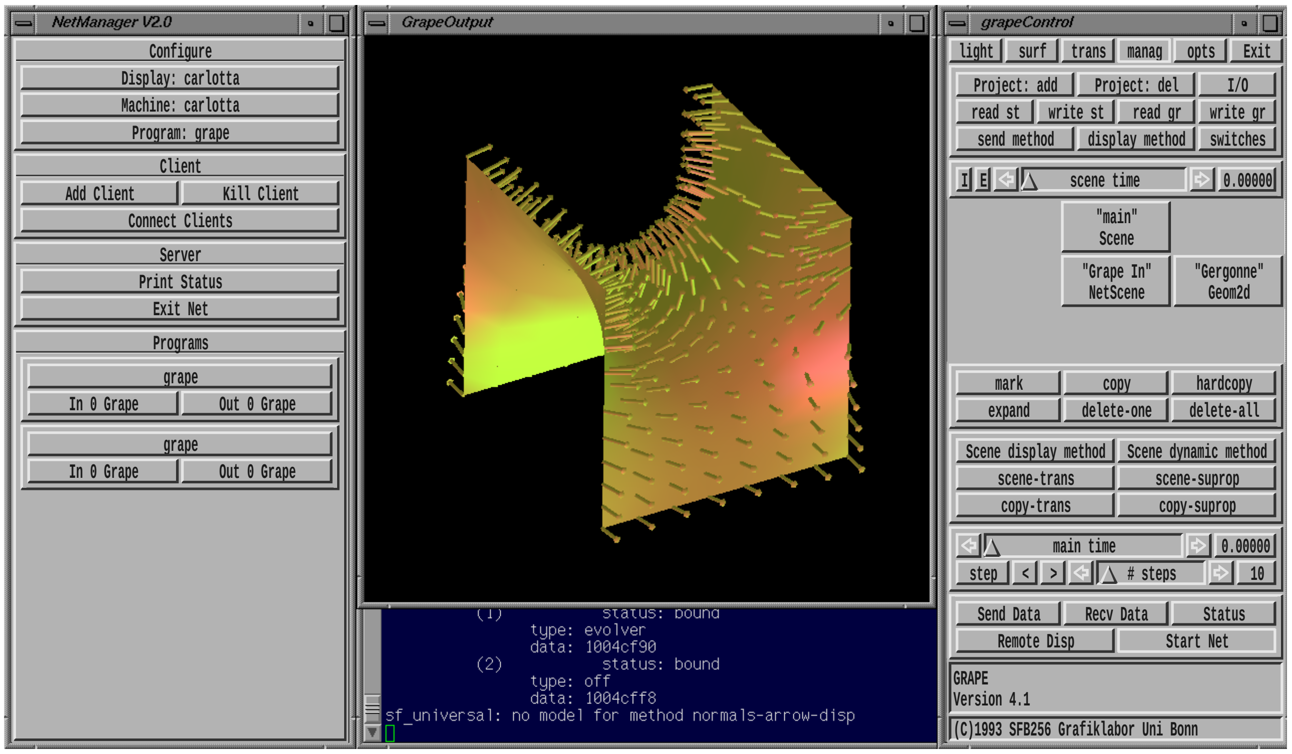
[Geo95] M. Phillips et al: *The Geomview Manual*, The Geometry Center, Minneapolis, 1995

[Gra95] *GRAPE User's Guide and Reference Manual 5.0*, Sfb 256 Universität Bonn, September 1995

[OP94] B. Oberknapp, K. Polthier: The Net-Manager Concept — Working Together and Distributed Computing, *GRAPE Newsletter 2,* Sfb 256 Universität Bonn, June 1994

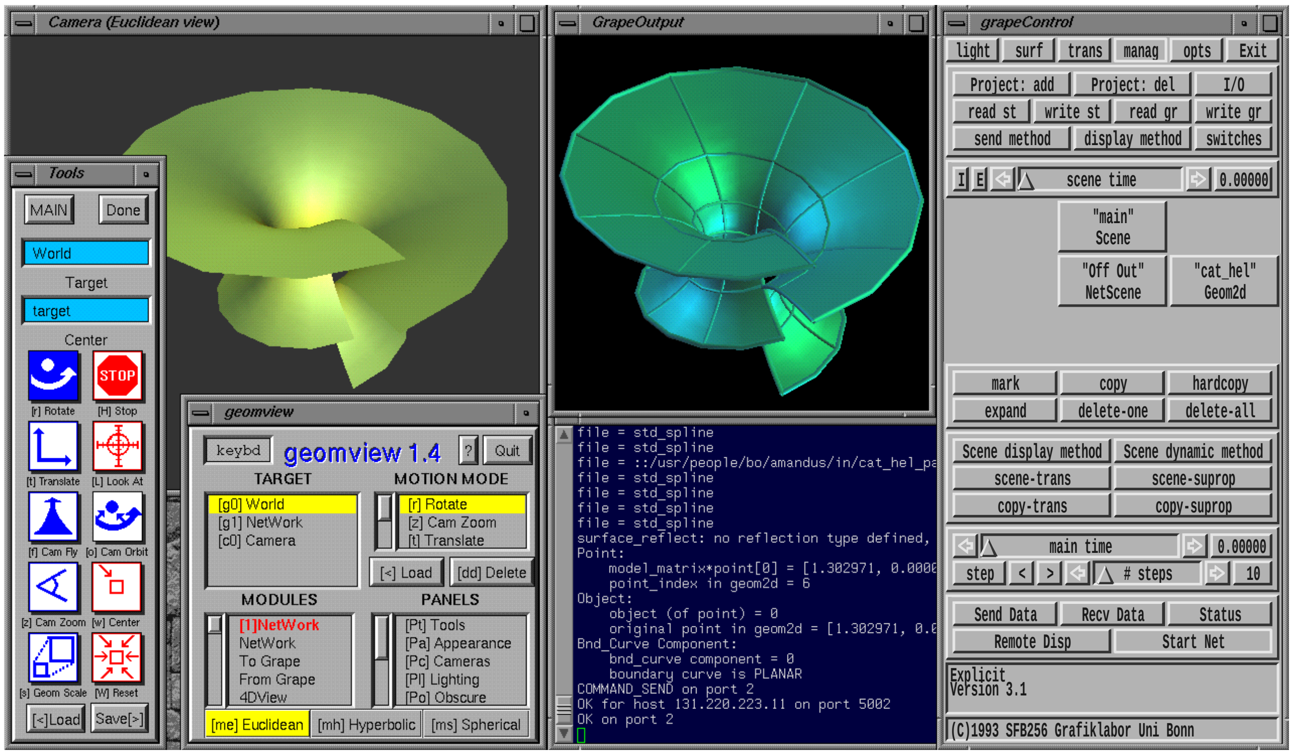
[PVM94] A. Geist et al: *PVM 3 User's Guide and Reference Manual,* ORNL/TM-12187, Oak Ridge National Laboratory, May 1994

[XDR] eXternal Data Representation, *Sun Technical Notes (for RPC 4.0),* Mountain View, California, Sun Microsystems, Inc.



```
        (1)              status: bound
                 type: evolver
                 data: 1004cf90
        (2)              status: bound
                 type: off
                 data: 1004cff8
sf_universal: no model for method normals-arrow-disp
```

Figure 3 (B. Oberknapp, K. Polthier: A Simple Concept for Distributed Computing in Computer Graphics): Remote Objects in GRAPE. The computation process is running on an SGI Challenge, the graphical postprocessing is done on an SGI Indigo2. The same geometry is accessed in the data hierarchies of both GRAPE processes.



```
file = std_spline
file = std_spline
file = ::/usr/people/bo/amandus/in/cat_hel_pa
file = std_spline
file = std_spline
file = std_spline
file = std_spline
surface_reflect: no reflection type defined,
Point:
    model_matrix*point[0] = [1.302971, 0.0000
    point_index in geom2d = 6
Object:
    object (of point) = 0
    original point in geom2d = [1.302971, 0.0
Bnd_Curve Component:
    bnd_curve component = 0
    boundary curve is PLANAR
COMMAND_SEND on port 2
OK for host 131.220.223.11 on port 5002
OK on port 2
```

Figure 6 (B. Oberknapp, K. Polthier: A Simple Concept for Distributed Computing in Computer Graphics): Geomview and GRAPE. An external Geomview module was used for connecting Geomview to the net-manager.