Virtual Laboratory: an interactive software environment for computer graphics

Pavol Federl and Przemyslaw Prusinkiewicz
Department of Computer Science, University of Calgary
Calgary, Alberta, Canada T2N 1N4
e−mail: federl|pwp@cpsc.ucalgary.ca

Abstract

Many activities in computer graphics can be regarded as experiments on virtual objects or models. In the process of experimentation the existing models are gradually improved and new model categories emerge. The Virtual Laboratory (*vlab*) is a software environment designed to support model development by facilitating the manipulation of models and providing mechanisms for storing and retrieving large numbers (e.g., thousands) of them. The models can be shared between users who work at different geographical locations over the Internet. In the paper we first clarify the essential concept of the Virtual Laboratory by describing its operation from a user's perspective. The modeling of plants serves as a sample application. We then present the key elements of *vlab* design and implementation, discuss the obtained results, and present their possible ramifications in the context of related ideas.

Reference

P. Federl and P. Prusinkiewicz: Virtual Laboratory: an Interactive Software Environment for Computer Graphics. In *Proceedings of Computer Graphics International 1999*, pp. 93−100.

Virtual Laboratory:
an Interactive Software Environment for Computer Graphics

Pavol Federl and Przemyslaw Prusinkiewicz
Department of Computer Science
The University of Calgary
Calgary, Alberta, Canada T2N 1N4
federl@cpsc.ucalgary.ca pwp@cpsc.ucalgary.ca

Abstract

*Many activities in computer graphics can be regarded as experiments on virtual objects or models. In the process of experimentation the existing models are gradually improved and new model categories emerge. The Virtual Laboratory (*vlab*) is a software environment designed to support model development by facilitating the manipulation of models and providing mechanisms for storing and retrieving large numbers (e.g., thousands) of them. The models can be shared between users who work at different geographical locations over the Internet. In the paper we first clarify the essential concept of the Virtual Laboratory by describing its operation from a user's perspective. The modeling of plants serves as a sample application. We then present the key elements of* vlab *design and implementation, discuss the obtained results, and present their possible ramifications in the context of related ideas.*

Keywords: interactive graphics environment, software design, graphical browser, object, inheritance, hyperobject, link, modeling, simulation.

1. Introduction

The Virtual Laboratory (*vlab*) is a software environment that originated from the need to organize and facilitate simulated experiments in computer graphics. The experimentation with visual models of natural phenomena is the focal application, but *vlab* has also been used to support experiments in fractal geometry and in physically-based modeling. In spite of differences in content, these activities share many common characteristics:

- An individual model can be conceptualized as a logically connected set of data files and programs that operate on these files.
- Model development and refinement is an incremental process in which small changes are made and programs are rerun many times. Differences between models are often small.
- The set (data base) of models grows continuously as new models are being created. The number of files included in this data base tends to be large (of the order of one thousand in our data base to date).
- Programs associated with the models may exist in many versions, and be invoked with a wide range of options. Only some versions and options are appropriate in the context of a specific object.

These characteristics make it difficult to organize work. At the level of individual models, experimentation is a tedious process, in which potentially useful modifications and improvements may be easily left unnoticed, or optimal parameters found and lost. Once the number of files produced during the experimentation increases, it is difficult to keep track of which files belong together, which programs and program versions are to be applied to them, and how to retrieve specific files among hundreds of similar ones. These problems are aggravated when models are revisited after some time, or when they are shared by many people (for example, in the context of collaborative work or computer-assisted instruction). The objective of the Virtual Laboratory is to help organize the work better by extending the rudimentary support offered by the underlying operating system (UNIX and its versions, IRIX, SunOS and Linux, in our current implementations).

The key to the organization of *vlab* is the observation that a model can be conceptualized as an object in the object-oriented programming sense, with files representing data and programs representing methods. In this context, *vlab* provides the following functionality:

File system organization

- Files constituting an object are grouped together for easy retrieval and access.
- The information about programs, program versions, and options applicable to specific objects is stored with the objects. Consequently, it is easy to experiment further on older objects or objects developed by others.
- New objects are easily created by the user and added to the data base. To save storage space, objects are saved incrementally, (*i.e.*, only the files that differ from the previous version are saved explicitly).
- Alternative arrangements of the same set of objects can be created by the user as structures of hyperobjects with pointers to the actual objects.
- Objects and hyperobjects may incorporate textual descriptions, presenting features inherent in an object or characterizing its occurrence in a particular hyperobject structure.

Support for interaction

- For the purpose of experimentation, the objects are automatically transferred to a temporary location, where they can be manipulated without fear that the original objects will be inadvertently changed or lost.
- Any program associated with an object can be added by the user to the object's menu. This menu provides a convenient method for initiating various tasks that constitute an experiment, facilitates re-running programs during incremental object development, and makes it possible to call programs quickly and without mistakes during interactive presentations.
- User-configurable control panels are included in the *vlab* environment to facilitate the manipulation of object parameters.
- A graphical browser makes it possible to visualize and manipulate the data base of objects, navigate through it, and access the objects.
- A hyperbrowser is provided to create and access structures of hyperobjects.
- The browser and hyperbrowser can be used to access remote *vlab* data bases across the Internet.

In the next section we describe previous work that provided the foundation for the Virtual Laboratory system presented in this paper. We then clarify the concept of *vlab* by describing its operation from a user's perspective (Section 3), and complement this description with a presentation of *vlab* design aspects less visible to the end-user (Section 4). The obtained results are discussed in Section 5, followed with conclusions in Section 6.

2. Previous work

Work on the Virtual Laboratory began in 1989, motivated by the need to organize a fast-growing data base of experiments related to plant modeling and fractal generation using L-systems [10]. The original concept and design were introduced and related to previous research results by Mercer, Prusinkiewicz and Hanan [7], and detailed in Mercer's M.Sc. thesis [6]. This work introduced several key elements of *vlab* design that remained essential to the subsequent implementations and extensions:

- Related data files were grouped into *objects*, complete with a *specification file* detailing which *tools* (programs with options and argument files) apply to this object.
- The objects were organized into a hierarchical data base called the *object-oriented file system* (*oofs*), governed by the *prototype-extension* relation between objects introduced by Lieberman [4].
- To prevent unwanted modifications, the objects were fetched from the data base to a temporary location called the *lab table* for experimentation.
- The user could configure virtual *control panels* to manipulate chosen model parameters.

Mercer's implementation included the first versions of the *browser* for navigating through the data base, the *object manager* for applying tools to objects, and the *control panel manager* for creating control panels. The browser displayed only a limited view of the data base (current object and its immediate extensions) and did not provide adequate support for moving objects within the hierarchy. A prototype graphical browser that addressed these limitations was developed in Tcl/Tk [9] by Lowe [5], using the tree widget implemented by Brighton (see [3]) as the point of departure. The underlying algorithm for visualizing tree structures was described by Moen [8]. Direct experience and a usability study performed by Lowe confirmed the convenience of browsing and reorganizing the *vlab* data base using its tree representation. Subsequently, Federl extended *vlab* with the capabilities for creating and presenting alternative views of the data base using the *hyperbrowser*, and for accessing remote data bases over the Internet [1]. We find the resulting system very useful in practical applications, which motivates our presentation of its present version in this paper.

3. A user's perspective of vlab

A sample screen of a Silicon Graphics workstation running *vlab* is shown in Color Plate 1. The windows on the left side belong to the browser and the hyperbrowser. The small windows along the top edge of the screen belong to the object manager processes, and represent objects copied to the *lab table*. The menu pulled down shows tools associated with the object *congo.demo*. The central part of the screen contains a window created by one of the tools, an L-system-based plant modeling program. Another tool is a control panel for manipulating model parameters, shown on the right side of the screen.

3.1. The browser

Typically, the user enters *vlab* by invoking the browser, which makes it possible to navigate through the objects in the lab by following the hierarchy induced by the prototype-extension relation (Section 4.1). Each object is represented in the browser's window by a folder symbol, an object name, and an optional icon. Multiple folders represent objects that have extensions, whereas single folders represent terminal nodes (leaves) of the hierarchy. Prototypes and extensions are connected by lines (Color Plate 1).

Since the number of objects in the data base may be large, the user can focus on its parts by dynamically expanding or contracting the tree representation of the hierarchy. Initially, the browser presents the root and the first level of objects. By clicking the name of an object the user causes the browser to display its immediate extensions. This makes it possible to expand a tree branch level-by-level. Alternatively, the user can apply a menu option to expand an entire subtree of extensions that originates from a selected object. In a similar manner, parts of the tree that are no longer of interest can be hidden from view.

In order to provide additional information about the objects, the user can display their icons, either individually (toggle action of a mouse button) or for an entire subtree (using a menu option).

In addition to browsing, the user can search for an object given its name. When the matched object is found, the hierarchy is automatically expanded and the browser's view is adjusted to include the object in the window.

The browser also makes it possible to reorganize the hierarchy of objects by renaming and deleting them, as well as cutting, copying and pasting to another location. Individual objects can be dragged from the source location to a destination with the mouse. Operations on entire subtrees require proper menu selections. To facilitate transfers between distant locations in the graphical view of the data base, the user may open several instances of the browser and copy and paste objects between them.

Once an object of interest has been located, the user can invoke it by clicking its file folder symbol or using a menu. This calls the object manager program, discussed next.

3.2. The object manager

The first task of the object manager is to copy the files constituting the chosen object from the data base to a temporary location for the purpose of experimentation. Maintaining the laboratory metaphor, we say that the object is copied from the *storage shelf* to the *lab table*. Next, the object manager creates a window with the object name and icon. By clicking this icon, the user pulls down a menu of tools, or programs associated with the object.

Let us consider the refinement of the plant model shown in Color Plate 1 as a typical example of the user's interaction with a *vlab* object. The user calls a simulation program from the object's menu. Another menu item is used to invoke a control panel manager. The panel acts as an editor of the object's data files [7]. Consequently, the current state of the model is always reflected in these files (in addition to the internal state of the active programs) and can be easily saved. Once the modifications have been completed, the user may apply an icon-making tool to capture a part of the screen as the new object icon. The object on the lab table can be saved either by overwriting the original object in the data base, or by creating an extension to it. The new extension is automatically added to the data base of objects and appears in the browser's window.

To serve as a general-purpose environment for simulated graphics experiments, the tools associated with an object and the files they operate on must be easily definable by the user. To this end, each object contains a *specification file* that lists the available menu items and defines the action resulting from each item's selection. For example, the specification file for the *congo.demo* object in Color Plate 1 includes the following lines:

```
generate:
    cpfg -s 300000 lilac.l lilac.v lilac.a
L-system:
    panel:
        panel panel.l | ped lilac.l
    EDIT lilac.l
```

Menu items end with a colon and may be nested to create a hierarchy of menus. The associated actions are specified using the syntax of UNIX commands. For instance, the menu item *generate* will invoke the simulation program `cpfg` with options `-s 300000` and three input files `lilac.l`, `lilac.v` and `lilac.a`. Similarly, the menu sub-item *panel* associated with the item *L-system* will invoke the `panel` program. Its appearance on the screen will

be defined by the input file `panel.l`. The results of control manipulation will be passed to the program `ped`, which will edit the corresponding parameters in the file `lilac.l`. The user can easily define or redefine the menu items or the operations associated with them by editing the specification file and rereading the modified file into the object manager.

The specification file may also include references to generic tools, illustrated in the example by the line `EDIT lilac.l`. The (sub)menu and the associated action are then defined in a system-wide file *tools*. For instance, `EDIT` may be defined there as follows:

```
EDIT
edit:
    jot
```

This indirect tool specification makes it possible to avoid repetitive definition of frequently used menu items and makes it easy to globally replace one tool by another (for example, `emacs` or `vi` by `jot`) in all objects.

In addition to the user-definable items, the menu displayed by the object manager always includes a predefined set of utilities, needed to:

- edit the specification file,
- save a portion of the screen as the object's icon,
- save the object by overwriting its storage location or by creating an extension to it,
- request the browser to highlight an object's position in the hierarchy (useful when there are several objects on the lab table, and the user needs to verify their positions),
- open a UNIX shell on the object's storage or lab table location to facilitate operations not supported by *vlab* (for example, construction of a new object prototype from scratch).
- quit the object manager.

3.3. The hyperbrowser

An object in the data base may be of interest in several contexts. For instance, the model shown in Color Plate 1 was developed as a part of a comparative study of lilac flowers for horticultural purposes, but it may also serve as an example of a particular branching architecture, an illustration of model construction according to field data, an instructional example of programming using L-systems, a realistic model available for incorporation in complex scenes, and the source of an image for a paper. A *vlab* program called the *hyperbrowser* allows the user to create alternative views of the object data base, reflecting conceptual associations between the objects rather than the default prototype-extension relationships. The hyperbrowser is manifested on the screen in a manner similar to the browser: as a window displaying a hierarchy of objects. (To avoid confusion, the user may customize the hyperbrowser differently than the browser, for example by changing background color and the icon size.) The objects are not added to this hierarchy automatically each time a new extension is created. Instead, they must be positioned explicitly by dragging from the browser window, or from a another location in the hyperbrowser window. To highlight the role of a particular object occurrence, the user may associate with it a textual file, which can be edited or displayed using the hyperbrowser's menu. Thus, an object occurrence in the hyperbrowser hierarchy can be thought of as a *hyperobject* consisting of a link to a real object and a text file.

The hyperbrowser also makes it possible to sequentially access objects with a common parent. This option is particularly useful when the Virtual Laboratory is used for interactive presentations.

3.4. Access to remote data bases

The Virtual Laboratory is not confined to individual machines and local file systems. The user may also invoke browsers or hyperbrowsers on different data base hierarchies, on the same local area network or over the Internet. Most operations available locally can be also performed on remote data bases, provided that proper permissions are given by their owners. For example, the user may browse a remote data base of objects, experiment with a remote object (which is transferred to the *local* lab table for fast experimentation), or copy objects and object hierarchies between various data bases. These transfer operations are performed by dragging and dropping, or copying and pasting, between browsers open on different data bases.

4. System design

In the previous section, we presented Virtual Laboratory components from the end-user's perspective. We will now describe the design that combines these components into a coherent system.

In order to see the reasoning behind some of the design decisions, let us complement the general objectives of *vlab* given in Section 1 with the additional requirements that guided the design.

- Individual *vlab* components should have clearly specified functions, so that they can be easily replaced by improved or redesigned components without requiring fundamental changes to the underlying data bases. An

analogy can be made with World Wide Web browsers, which are being improved and extended without requiring changes to the Web.

- To allow for incremental development of *vlab*, users should be able to perform functions directly from UNIX if they are not (yet) supported by *vlab*.
- The Virtual Laboratory should provide a convenient framework for the use of a wide range of application programs. Consequently, as few assumptions as possible should be made regarding the operation of these programs.

The essential elements of *vlab* design are described below.

4.1. The object-oriented file system

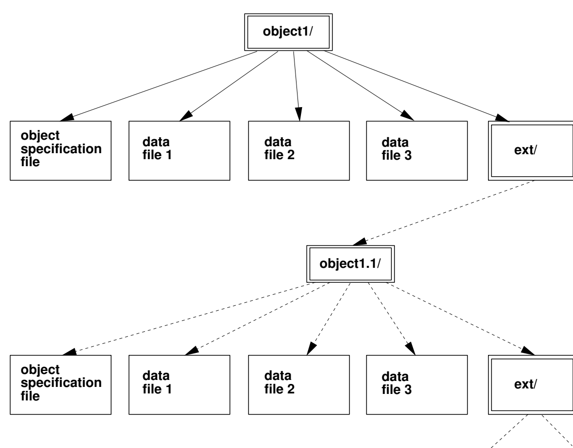

Figure 1. Structure of the object-oriented file system. From [7].

A laboratory *object* is defined as a directory (with the name corresponding to the object's name) that contains:

- the *data files* that comprise a particular model,
- the *specification file* (Section 3.2),
- the object's *icon*,
- the *identification file*, which holds the object's *identification number* (*id*) for linking with hyperobjects (Section 4.2),
- an optional subdirectory of object *extensions*.

The objects are organized into an *object oriented file system* (`oofs`), which constitutes the main data base of objects or experiments. Its definition is compatible with the UNIX

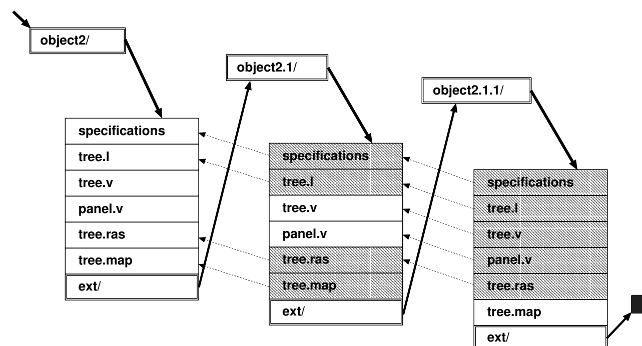

Figure 2. A prototype with a sequence of extensions. Shaded areas indicate links. From [7].

file system to facilitate operations not supported directly by *vlab*, such as construction of new prototypes from scratch and construction of objects that share files with several other objects. This data base can be represented by a hierarchy of UNIX directories and files, as shown in Figure 1.

The path of subdirectories leading to an object establishes the inheritance structure of the lab. Inheritance is based on the notion of specifying a new object in relation to an existing parent object [4]. The old object is the *prototype* and the new one is its *extension*. The extension contains only those files that are different from the corresponding files in the prototype. Files that remain the same are inherited from the prototype using symbolic links. In other words, an object in the data base contains those files that are unique to the object, and links to files that are inherited from its prototype (Figure 2). This approach saves space and allows a single change in the prototype to propagate through all extensions.

An object on the lab table differs from its data base counterpart in two respects: the symbolic links are replaced by actual files, and the extensions directory is not present. When a new extension is created, the files on the lab table are compared with those in the prototype object; those files that differ from the prototype are saved, and links to the remaining files are established automatically.

A similar process is observed when an object is copied from one location to another. The object is first copied to a temporary location, then pasted to the new one. During the paste operation, the browser compares the object's files with those in the new prototype, and replaces repeated files by symbolic links. Thus, objects can be relocated without introducing inconsistencies into the data base.

The browser supports two modes of the copy-and-paste operation, called *links stay* and *links move*. In the first case, the objects being copied retain their *id* numbers, and the objects being pasted are assigned new numbers. In the sec-

ond case the situation is reversed: the original objects are assigned new *id* numbers, while the old numbers are transferred to the pasted objects. The role of these modes will be explained in the next section.

4.2. The hyperobject file system

Conceptual relations between objects are represented by the structure of a *hyperobject file system* (*hofs*). Similarly to the *oofs*, the *hofs* is a hierarchy of UNIX directories and files grouped into fundamental units, *hyperobjects*. A hyperobject directory includes:

- The *node* file, containing the *id* of the corresponding object and a list of hyperobject extensions. This list specifies the order in which the extensions will be displayed by the hyperbrowser, which is important when the objects are accessed sequentially (Section 3.3).
- A text file intended to contain a description of object features pertinent to its specific occurrence in the *hofs*;
- A subdirectory of hyperobject extensions.

The node numbers introduce a level of indirection to object references, which simplifies the maintenance of links when the location of target objects in the *oofs* is changed. Specifically, the two modes of copy-and-paste operation allow the users to decide whether links from the hyperobjects should remain with the original objects, or transfer to the pasted ones. Quick access to objects from hyperobjects is made possible by an *object reference table*, which specifies the path to each object given its *id*. This table is updated by the browser in response to the user's actions affecting the object data base.

4.3. The vlab daemon

In order to operate in concert and present the user with a consistent view of the *vlab* state, *vlab* processes must communicate with each other. For example, the creation of a new object extension or the deletion of an object cause all active browsers that display the affected part of the data base to update their windows. The interprocess communication is implemented using the *vlab daemon* (Figure 3). Each process has a separate communication channel to the daemon, implemented using sockets. A new process registers with the daemon and indicates which message types it is interested in receiving. The daemon transparently forwards any message sent by an active *vlab* component to all interested receivers. The resulting star configuration simplifies message routing compared to direct bilateral communication between pairs of processes. In addition to support for communication, the daemon launches selected objects (i.e. invokes the object manager) upon requests from a browser

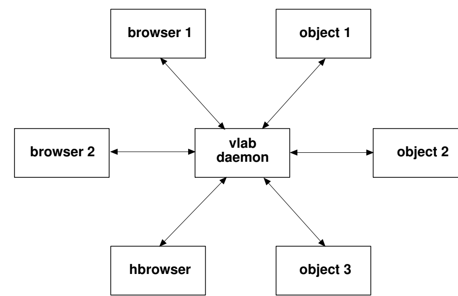

Figure 3. Communication between vlab system components.

or a hyperbrowser. The daemon is invoked automatically and thus remains invisible to the user.

4.4. The remote access server

Access to remote data bases is accomplished using the *remote access server*, which runs as a daemon on a remote machine, and performs operations on behalf of the client browser. The underlying mechanism can be viewed as a simplification of UNIX Remote Procedure Calls (RPC) [11]. The format of messages exchanged between the browser and the remote access server has been optimized to reduce response times. For example, the browser can ask the server for information regarding an entire subhierarchy of objects in a single message, and this information is also returned in a single message.

4.5. Vlab maintenance

In addition to the programs described so far, *vlab* includes utilities that check the integrity of the data base (*oofs*, *hofs*, and the object reference table), and assist in its repairs. Possible inconsistencies include symbolic links to nonexisting files in the *oofs*, incorrectly specified object location in the object reference table, and reference to a nonexisting object from a hyperobject. Typical sources of errors are system or network connection failure during an operation, and incorrect object manipulation by the user at the UNIX shell level.

5. Implementation and results

The Virtual Laboratory has been developed on SGI machines using OpenGL and Motif libraries. It has been ported to SUN workstations (under SunOS), and Linux. The source (in C and C++) has approximately 45,000 lines of code. Our local data base has approximately 1,000 objects comprising 10,000 files.

Table 1. Time needed to perform selected *vlab* operations on a 150MHz/32MB SGI Indy R5000 workstation, using local disk.

Operation	Number of objects	Time[a] [s]
Expand subtree	10	instantaneous
	100	1
Show all icons (60×60 pixels)	10	1
	100	7
Copy[b]	10	3
	100	21
Paste[b]	10	6
	100	56
Drag and drop[c]	1	1
Get object[c]	1	1
New extension[c]	1	1
New link[d]	1	instantaneous

[a] Average from 5 measurements
[b] Average object size: 25 KB
[c] Object size: 100 KB
[d] Any object size

The speed of *vlab* operation is determined primarily by the access time to the *oofs*, thus it depends on whether *oofs* is mounted locally or accessed over a network, and the network's speed. For orientation purposes, some measurements describing *vlab* performance have been collected in Table 1.

Vlab has been used to support an ongoing collaborative research program in plant modeling and visualization, which has now branched to several locations in North America, Australia, and Europe. Approximately 15 people, both computer scientists and biologists, have used versions of *vlab* for up to eight years. Our collective experience is summarized below.

5.1. Advantages

Vlab is an essential element of our software environment. It makes it easy to experiment with the models, retrieve and resume work initiated in the past, and organize the results for publications and presentations.

The possibility of modifying objects without fear of losing the original data, menu-driven access to the object tools, and object manipulation with virtual control panels are the keys to successful experimentation by novice and experienced users alike. The flexible association of objects into structures accessed using the hyperbrowser makes it possible to use *vlab* as an attractive vehicle for interactive presentations. We also found hyperobject structures useful in the conceptualization of the relationships between the objects, although we do not yet have long-term experience with this aspect. The remote-access capabilities have been essential in collaborative work involving researchers at different locations. Remote access is also important to users who operate on the same data base from different locations, for example at work and at home. Finally, we have developed objects belonging to various domains, from the visualization of algorithms to fractals and tilings and from cellular automata to physically-based models of mechanical systems, which supports the claim to the versatility of *vlab* applications. For example, Color Plate 2 shows a physically-based cloth model that was organized as a *vlab* object. In this case, none of the domain-specific files or programs (the simulation program, its data files, even the control panel) is the same as for the plant models. Nevertheless, the general functions provided by *vlab* — the access to the object using the browser or the hyperbrowser, and the possibility of manipulating it using the object manager — remain the same.

The first implementation of *vlab* was created in 1990. Since then, the original programs have been redesigned and reimplemented, and new components and features have been introduced. In spite of these changes, we were able to maintain continuity of our data base over the past eight years. This attests to the viability of the foundations of the *vlab* design, in particular the concept of objects with associated specification files, organized into an object-oriented file system with inheritance based on the prototype-extensions relation. In summary, the Virtual Laboratory meets the objectives stated in Section 1.

5.2. Limitations

Experience with *vlab* also brought to attention some limitations of the present design. The most important is the lack of support for hierarchical model construction. For example, the user should be able to assemble a forest scene from objects representing individual plants, which in turn would be constructed from objects representing organs: leaves, flowers, or fruits. Unfortunately, *vlab* does not currently provide a mechanism for constructing objects that include other objects as components, or inherit files from several prototypes.

Vlab also does not produce visual cues indicating which application program has been spawned by which object manager. This may lead to confusion when the same tool is applied to several objects lying on the lab table at the same time.

Objects may contain files that differ drastically in size. When objects are accessed remotely over a slow communication link, it would be convenient to exclude large files that can be easily reconstructed locally, such as image files representing the results of short simulations. At present, *vlab*

does not allow for a selective transfer of parts of objects.

In the case of a long simulation, it would be convenient for the user to exit *vlab* with the simulation running in the background, and resume *vlab* operation after some time, with the object manager reopen automatically on the object left on the lab table. This features is not yet provided.

5.3. Design alternatives

Our implementation of the object oriented file system was significantly affected by the requirement that the objects should be easily accessible by the user directly from the UNIX. As our confidence in *vlab* increases, this requirement is becoming less important, which opens the way for alternative implementations of the *oofs* data base. Minor modifications are:

- Inclusion of the object extension directories into the prototype objects, without the intermediate *ext* directories shown in Figures 1 and 2. This would reduce the length of object pathnames, whereas object directories would contain a mixture of object files and extension directories.

- Removal of all symbolic links. A file listed in a specification file but absent in the object directory would be automatically inherited from the closest prototype that has it. This would eliminate the proliferation of symbolic links inherent in the current design of the *oofs*. On the other hand, the user would be practically unable to access files making up an object without the assistance of *vlab*.

A more drastic change stems from the realization that a hierarchy of hyperobjects is similar to the UNIX directory structure. Pursuing this analogy, we could design a *vlab* data base in which all objects are placed in one UNIX directory, and are accessed exclusively through links from the "directory structures" maintained by the browser and the hyperbrowser. This would offer the following benefits:

- The browser and the hyperbrowser would access objects in a unified manner, which would simplify their design.

- Changes in the inheritance structure would be implemented by rearranging links to objects, and inserting or deleting selected files as required by changes in object prototypes. The positions of objects in the underlying UNIX file system would remain intact. The objects could be therefore identified directly by their UNIX pathnames, eliminating the need for the object identification numbers and the object reference table.

6. A concluding remark

Our long experience with the consecutive incarnations of *vlab* shows that it provides an effective and pleasant environment for organizing and conducting interactive experiments with visual simulation models. Gentner and Nielson [2] envisioned that future computer users will focus on manipulating huge numbers of complex information objects while being connected to a network shared by other users and computers. This is exactly what *vlab* makes it possible to do.

7. Acknowledgments

We would like to thank Lynn Mercer for continuing discussions of the design directions being embedded in the vlab, and for her comments on this paper. We would also like to thank dedicated vlab users, Mark Hammel, Jim Hanan and Radomír Měch, for comments that led to the refinement of its design, and merciless testing.

This work has been sponsored by research, equipment, and infrastructure grants as well as postgraduate fellowships from the Natural Sciences and Engineering Research Council of Canada, and a Killam Resident Fellowship.

References

[1] P. Federl. Design and implementation of Global Virtual Laboratory — a network-accessible simulation environment. Master's thesis, University of Calgary, 1997.

[2] D. Gentner and J. Nielson. The Anti-Mac interface. *Communications of the ACM*, 39(8):70–82, 1996.

[3] M. Harrison. *Tcl/Tk tools*. O'Reilly and Associates, 1997.

[4] H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 214–223, New York, 1986. Association for Computing Machinery.

[5] E. M. Lowe. Extensions to the Virtual Laboratory. Master's thesis, University of Calgary, 1995.

[6] L. Mercer. The virtual laboratory. Master's thesis, University of Regina, 1991.

[7] L. Mercer, P. Prusinkiewicz, and J. Hanan. The concept and design of a virtual laboratory. In *Proceedings of Graphics Interface '90*, pages 149–155. CIPS, 1990.

[8] S. Moen. Drawing dynamic trees. *IEEE Software*, pages 21–28, July 1990.

[9] J. K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley, 1994.

[10] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[11] W. R. Stevens. *UNIX network programming*. Prentice-Hall, Englewood Cliffs, 1990.

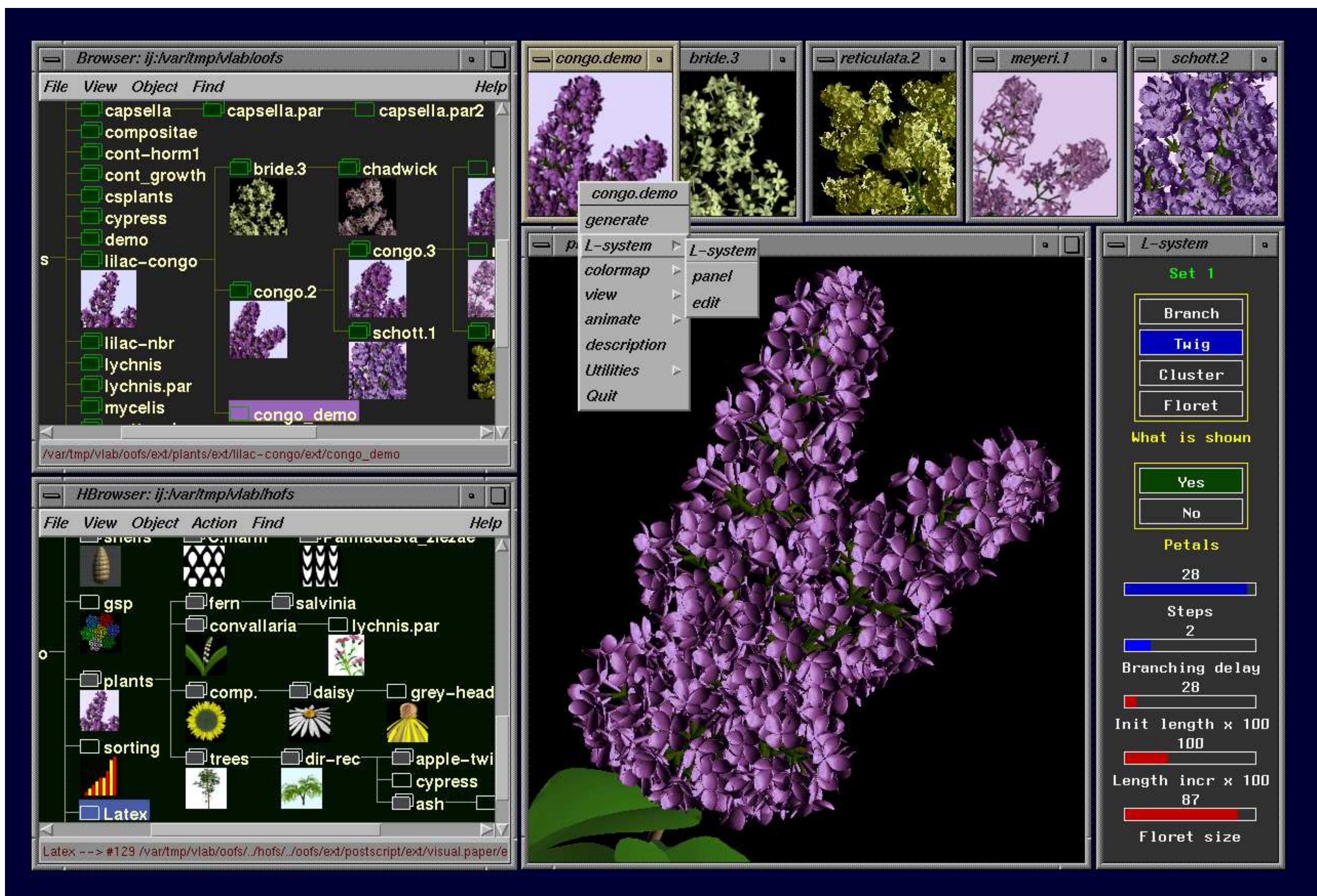

Figure 4. A sample vlab screen

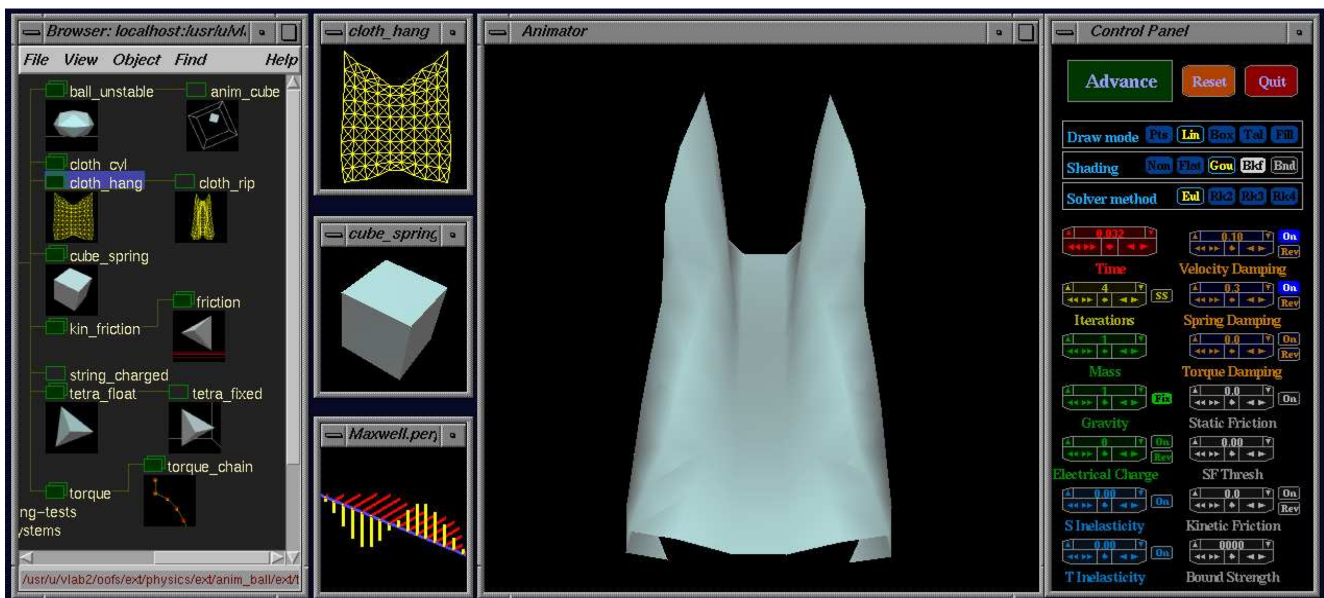

Figure 5. A physically-based experiment carried out in the vlab environment