Visualization of Geometric Algorithms in an Electronic Classroom *

Maria Shneerson
Department of Applied Mathematics
The Weizmann Institute of Science
Rehovot, Israel, 76100

Ayellet Tal
Department of Electrical Engineering
Technion – Israel Institute of Technology
Haifa, Israel, 32000

Abstract

This paper investigates the visualization and animation of geometric computing in a distributed electronic classroom. We show how focusing in a well-defined domain makes it possible to develop a compact system that is accessible to even naive users. We present a conceptual model and a system, GASP-II, that realizes this model in the geometric domain. The system allows the presentation and interactive exploration of 3-dimensional geometric algorithms over the network.

Key words: Algorithm animation, Visualization in Education, Geometric algorithms.

1 Introduction

A large part of computer science education deals with algorithms and data-structures. Though algorithms are dynamic in nature, most instructors choose static ways to describe them. It has been shown that visualization can be a powerful tool in the teaching of algorithms [7]. An algorithm animation can expose properties of an algorithm that are otherwise hard to grasp, and help get some intuition into the way the algorithm operates. The lack of adequate software tools is one of the major reasons that visualization is rarely being used in class.

We focus on a particular domain, that of computational geometry, where the algorithms can be highly complex and difficult to grasp. The ability to follow the manipulation of geometric objects all the way can greatly facilitate the understanding of an algorithm. Unfortunately, there are few visualization tools available and even fewer which exploit the visual nature of geometric algorithms.

Most of the research in the area of algorithm animation aims at developing general-purpose systems, which are designed for fundamental algorithms (e.g., Balsa [5], Balsa-II [2], Tango [10], Zeus [3], CAT [4]). Their support for 3D geometric algorithms is limited. There exist a few special purpose systems for geometric computing that come to ease the unusual difficulty of the implementation cycle (e.g., [8], [9], [6]). The Computational Geometry Workbench [6] and the XYZ GeoBench [9] support 2D algorithm animations, but this is not their chief goal, and thus creating an animation is still not trivial. Mocha [1] provides 2D algorithm animations over the World Wide Web. GASP [13] goes beyond these systems by providing a rich set of 3D visualization and animation tools.

We describe in this paper our system, GASP-II, and present the model that underlies its design. GASP-II allows the presentation and exploration of 3D geometric algorithms in a distributed electronic classroom. Views of the same running algorithm may reside on different machines, that can be distributed in various geographic locations. While in most other electronic classroom systems the students are completely passive and the system simply displays the same window on multiple machines, GASP-II allows the students to actively participate in the process of learning by interacting with the algorithm and collaborating with the instructor.

An important capability of the system, is that during a lecture each student can observe a completely different view of the same running animation. All the student needs to do is to change a few parameters in a *style panel*, without any need to modify any code. GASP-II also provides every student with a control panel that lets the student not only control the speed of the animation but also rewind and rerun it, until the difficult parts of the algorithm are fully understood. This mode of studying is similar to studying from an interactive electronic textbook. GASP-II also supports a mode of studying in which the student is more limited and has to follow the instructor in the course of the lecture.

The design of GASP-II allows each machine to be in charge of the graphical work displayed on it. One advantage of this architecture is that each student can have a full control over the animation. Another benefit is that potential network traffic bottlenecks are avoided since only a relatively small amount of short messages is sent over the network. Moreover, in contrast to the X approach, the least powerful machine does not determine the rate of the whole class.

The next section describes the conceptual model that underlies the design and the development of GASP-II. Section 3 describes the end–user perspective. Section 4 presents the programmer perspective. Section 5 discusses the style of an animation. Section 6 demonstrates how GASP-II can be used in the classroom. Section 7 discusses implementation issues. We conclude in section 8.

2 The Conceptual Model

We present in this section our conceptual model for creating an algorithm animation system for an electronic classroom, that focuses on a well–defined domain. We distinguish between two types of users – programmers who wish to design and create an animation, and end-users (students and instructors) who interact and experiment with a finished animation.

A major principle of our model is that programmers should concentrate on the contents of the animation (i.e., the geometric operations that build up the algorithm), rather than on its visual aspects. How the algorithm is visualized is left for the system to automatically determine. Automating the process of generating animations is possible in a restricted domain, where a system can embed knowledge about the types of objects and the kinds of operations which are typical to the domain, and the proper ways to visualize them.

Following the GASP framework, the programmer needs to annotated the algorithm with *atomic units*, where interesting events occur. Each atomic unit groups together a few geometric operations, and can be looked upon as representing a logical phase of the algorithm. Each atomic unit consists of one or more calls to functions in the system's library. The system should animate the operations enclosed within an atomic unit concurrently, as a single unit.

During class, the visualization system should support two modes of studying – *self-study* and *guided-study*. In the self-study mode,

*This work has been supported in part by the Israeli Ministry of Science, Eshkol Grant 0554-2-96.

the lecture becomes similar to studying a chapter from an interactive electronic textbook. In the guided-study mode, the students must follow the instructor throughout the lecture. It is up to the instructor to decide when, during the lecture, a self-study mode is appropriate and when the students should be guided by the instructor. In the latter case, the system should ensure that all the views are synchronized with the instructor's view.

The system should support an electronic discussion between the instructor and the students. Since an unlimited number of students should be able to view the same algorithm, and they need not be at the same geographical location, electronic discussions become vital. An electronic discussion is more than a text-exchange, since it is accompanied by a 3D animation of the algorithm, which illustrates the topic discussed.

End-users should be able to change the default animation created by the system. In this case, students can watch different views of the same running algorithm, in order to accommodate their personal taste and to enhance their understanding. The users should be able to easily select their preferred *style* of animation through a comfortable user interface, without any need to modify any code. The style should affect the animation, not the implementation. Moreover, there should be no need to program each graphical view beforehand. This is an important advantage of an algorithm animation system for a restricted domain over a general-purpose algorithm animation system. The system can have sufficient knowledge regarding its domain, and can know how to generate various *styles* of visualization for the major objects and operations in that domain. In other words, even if the style is modified, the animation should still be generated automatically by the system, only a different animation should be generated.

3 Exploring an Animation

This section describes the GASP-II's environment. While exploring an algorithm animation, end-users (students, instructors, and programmers) can view the algorithm animation, interact with it, get textual explanations, and discuss the algorithm.

Viewing the animation: The animation is observed in the *Algorithm Window* (on the right hand side of Plate 1(a)). Algorithm windows use Inventor's *Examiner-Viewer* and thus are decorated with thumbwheels and push buttons which allow the user to interact with the animation. In addition to the conventional buttons, the left-hand side push buttons give the user information about the state of the algorithm.

Interacting with the animation: The animation is controlled through the *Control Panel* (on the top left of Plate 1(a)). The control panel consists of *general controls* and *study-mode controls*. Through the VCR-like general controls, the viewer can run the algorithm at varying speeds, pause the animation at various points of its execution, and "rewind" the algorithm in order to observe the confusing parts of the algorithm multiple times. This is an invaluable tool both for education and for debug.

The instructor's special status is reflected in the study-mode controls. The instructor can choose between a self-study mode and a guided-study mode. In the latter case, the general controls at the students' screens are disabled, and the students must follow the instructor in the course of the lecture. The student's study-mode controls support two levels of synchronization. With *instructor unit* synchronization, the animation viewed by the student shows the same atomic unit of the animation as the instructor's. GASP-II executes this command by skipping over all the in-between frames without executing them. With *instructor home position* synchronization, the current camera positions on both screens are the same. The student can then observe the scene in the exact same way as the instructor.

Getting textual explanations: The *Text Window* (on the bottom left of Plate 1(a)) is used to elucidate the events and to direct the viewer's attention to specific details. Every atomic unit of the algorithm can be associated with a piece of text which explains the events occurring during this phase. When the current unit changes, the text in the window changes accordingly. Text windows are especially important when the instructor and the students are not at the same geographical location, and when the system is used as an electronic textbook.

Discussing the algorithm: GASP-II supports an electronic discussion between the instructor and the students. A student can ask a question by pressing the *Question* button on the control panel, which causes a text window to pop up, and lets the student edit the question. The instructor controls the electronic discussion through the control panel by using the question control buttons and the *Send text* button. The students' questions are being accumulated in a queue, and can be displayed upon the instructor's request by selecting the *Next question* button. When the student's question is presented to the instructor, it is being accompanied by the animation of the appropriate atomic unit. The instructor can then reply by sending additional text containing further explanations. The explanations, along with the related animation, can be sent either to the student who asked the question or to the whole class.

4 Creating an Animation

According to our conceptual model, programmers should only be concerned with the design of the building blocks the animation consists of, but not with any of the presentation aspects of the animation. Recall that an animation consists of *atomic units*, which groups together a few geometric operations in the GASP-II's library. This library provides a rich set of functions which enable the programmer to write animations with minimal effort. The library contains three types of functions, as described below. (See [12]) for the full list of functions.)

1. Functions which animate the creation, modification, and removal of geometric objects in 2D as well as in 3D (e.g., polyhedra, meshes, spheres, cylinders, cubes, cones, planes, lines, points). For example, for modifying polyhedra, the library provides functions for adding faces and vertices, removing faces and vertices, joining polyhedra, etc. Each such operation is animated smoothly in a suitable way.

2. Functions that smoothly animate the modification of combinatorial objects such as lists and trees. For example, subtrees can be added to a given tree or removed from it.

3. Motion functions that contain, in addition to affine transformations, also float operations (e.g., motion on a Bézier curve).

An important feature of our model is that no parameters of a graphical nature (e.g., the way each operation is animated and the way each object is visualized) are required. The application transfers to GASP-II only parameters which are relevant to the geometry (e.g., the vertices and the faces of a polyhedron). All the visual aspects of the animation are left as empty slots that GASP-II fills up.

5 Animation Style

Once the geometric operations that build up the animation have been specified by the programmer, GASP-II automatically creates a default animation. One of the most distinguishing features of GASP-II is that end-users can easily modify this default animation. This is done by changing a few parameters in the *Style Panel* (Plate 2), without any need to write code. Unlike other algorithm animation systems, with GASP-II different graphical views need not be programmed beforehand. The system can automatically generate many animations for the same algorithm.

Through the style panel, both *global features* and *local features* can be specified. Global features influence the look of the whole animation, while *local features* influence a specific atomic unit of the algorithm (Plate 2(a)). There are many properties that can be specified in this panel, among them the way each operation is animated, the way each object is visualized, the colors, the transparencies, the speed of the animation, the number of frames per atomic unit, etc.

The style of each operation can be modified by selecting the *Function Style* button (Plate 2(b)). GASP-II can produce a few styles of animation for each operation. For example, a polyhedron can be created by fading into the screen, by growing up from a point, or by traveling in. The way various objects appear on the screen can be modified by selecting the *Object Style* button. For instance, a polyhedron can be shown flat, smooth or wire-framed, its edges (or vertices) might be displayed or not.

It is worthwhile to notice that this scheme allows users at the low-end machines to specify style parameters that fit their machines. For instance, they can decide that less frames will be used during the animation in order to keep up with the rest of the class.

6 Case Study

This section describes how GASP-II might be used in order to create and present an animation of an algorithm for finding a correspondence between 3D polyhedra [11]. The algorithm first "inflates" the given polyhedra into convex polyhedra. Then, a correspondence between the boundaries of the convex polyhedra is found, and induced back onto the original objects. "Inflating" a polyhedron is done in two phases: simplification and re-attachment. The simplification is carried out by removing a low degree vertex from the vertex graph of the polyhedron, and re-triangulating the resulting hole. This operation is repeated until a 4-clique graph, representing a tetrahedron, results. At this point the process is "reversed" and the vertices are re-attached, this time in a convex fashion.

Plate 1 demonstrates one step in the simplification phase. During the animation, the selected vertex is shown in blue (Plate 1(a)). Its cone of faces smoothly drives away from the polyhedron. The animation is accompanied with text explaining the criterion for the selection of a "good" vertex. Next, the hole created is re-triangulated. Finally, the highlighted vertex and its cone of faces fade out. This procedure is repeated until a convex polyhedron results.

Exploring the Animation: Plate 1 shows three typical screens during a lecture on shape correspondence, taken at the same time. The instructor's screen, shown in Plate 1(a), displays one view of the algorithm, text which explains the current unit, and the instructor's control panel. The polyhedron displayed in the algorithm window has two colors. The original faces are colored red while the new faces (created thus far during the simplification phase) are colored blue. Plate 1(b) shows a student's screen. It displays a somewhat different view of the algorithm, the same text as the instructor's, and a student's control panel. The view is different since the student changed the colors and the transparency values in the style panel. The student's screen also contains a question this student is currently editing. Plate 3 animates a previous unit of the algorithm (where the first vertex is removed). This student rewinded the algorithm to a previous unit.

Creating the Animation: The simplification phase of the shape correspondence animation consists of three atomic units enclosed within a for–loop. The first unit removes a single vertex along with its cone of faces. The second unit adds the triangulation faces. The third unit removes the above cone of faces from the screen.

To generate the animation, all the programmer needs to do is to implement the short code, shown in Fig. 1. The code consists of calls to functions included in the GASP-II's library. Each atomic unit begins with a call to `GASP_Begin_atomic` and ends with a call to `GASP_End_atomic`. In between, there are calls to functions that generate the animation. Recall that all the operations within an atomic unit phrase are animated concurrently. Within the first atomic unit there are two operations. The function `GASP_Split_mesh` starts moving the specified vertices along with their cone of faces away from the polyhedron, creating black holes in it. Each such cone travels in the direction of the vector which is the difference between the vertex (which created the cone) and the center of the split polyhedron. The function `GASP_write_to_text_view` writes the explanation in the text-window. The second atomic unit calls the function `GASP_Add_faces` which creates the new triangulation faces by fading them into the scene. GASP-II gives the new faces different colors, so that they can be distinguished from the old faces. The third atomic unit calls the function `GASP_Remove_object`, to remove (by fading out) the vertex and its cone of faces from the scene. Note that no parameter of the graphical nature is transferred to GASP-II.

```
/* Remove the selected vertex */
GASP_Begin_atomic(split_atomic_name);
  GASP_Split_mesh(new_mesh_names, previous_mesh_name,
                  remove_vert_no, remove_vertices);
  GASP_Write_to_text_view("TEXT",
   "During the simplification phase, a low-degree vertex
    is removed and its vertex-graph is re-triangulated."
GASP_End_atomic();

/* Re-triangulate the hole */
GASP_Begin_atomic(add_atomic_name);
  GASP_Add_faces(new_mesh_names[0], face_no, faces);
GASP_End_atomic();

/* Remove the detached cone */
GASP_Begin_atomic(remove_atomic_name);
  GASP_Remove_object(new_mesh_names[1]);
GASP_End_atomic();
```

Figure 1: Simplification Animation Code

Animation Style: To generate the snapshot shown in Plate 1, the instructor needs to specify in the style panel that the original polyhedron should be red, and that the vertex removed and the new faces should appear in blue. As for the other properties of the animation (e.g., using fading for the creation and removal of faces, removing vertices by driving away their cones of faces from the center of the polyhedron), the defaults selected by GASP-II are used. To generate Plate 2, the above two colors need to be specified (as green and yellow, respectively), along with two transparency values. Finally, to generate Plate 3, all the GASP-II's defaults are used and no modifications in the style panel are required.

7 Implementation

Fig. 2 illustrates the GASP-II's architecture. The processes running on the students' machines are in charge of executing the animation and handling the viewer's input on these machines. They communicate with the instructor's machine through messages. Two processes are running on the instructor's machine. Process 1 consists of the application code and the calls to the functions contained in the GASP-II's library. Process 2 has two tasks. It is responsible for executing the animation and handling the viewer's input on the instructor's machine. In addition, it is in charge of sending instructions regarding the animation to the students' machines.

Suppose, for instance, that the application initiates a call to the `GASP_Create_mesh` function. This function, which belongs to Process 1, prepares a few messages containing the type of the operation (`Create_mesh`) and the relevant information for this operation (the vertices and the faces), and sends them to Process 2. Utilizing a hand-shaking approach, Process 1 then waits for a reply from Process 2. Note that the messages delivered contain only high-level

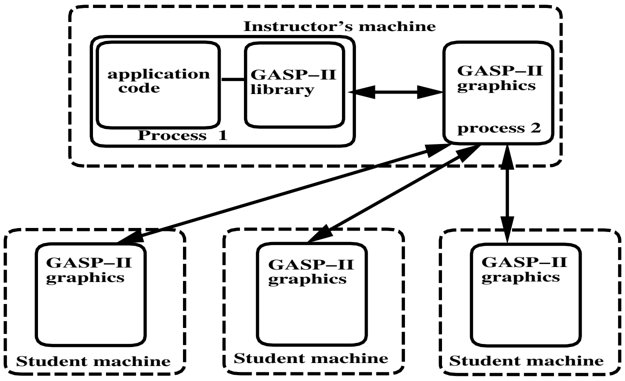


Figure 2: GASP-II's architecture

information, rather than the detailed graphics that needs to be carried out. Upon receiving the messages, Process 2 updates its internal data structures, consults its style information, and if necessary, executes the animation according to its style (e.g., fades in the mesh in a particular color over a specific number of frames). It also broadcasts the messages to the students' machines, but does not wait for replies from the students. Process 2 then sends an acknowledgment to Process 1. Process 1, which has been waiting for that reply, returns to the application's code. Only then can the application proceed executing the algorithm. Meanwhile, on each of the students' machines, the internal data structures are being updated, the style information is being consulted, and the animation is executed. The animation might be different from one user to another, depending on the user's specified style.

This architecture has several benefits. First, the graphical work, which is resource consuming, is distributed among all participants in the classroom. This frees up the instructor's machine to mostly be in charge of its own graphics. Second, this architecture allows the students to choose their own styles for a specific animation and view completely different animations of the same algorithm. This is so because the messages received by the students' machines do not specify how exactly should the animation be carried out, and thus each machine is free to determine the appropriate animation using its own style. Third, since the information sent is very high level, the network traffic is very low. Fourth, if synchronization is required, and explicitly requested by the instructor or by the students, it can be easily handled. Fifth, the less powerful machines do not slow down the class, since they work independently. Sixth, if a failure occurs at a student's machine, it does not affect the other machines which keep working as usual (since the instructor's machine does not wait for replies from the students' machines). Seventh, the system fully protects the algorithm code which is never sent over the network. (Yet, if required, it can be displayed on a text window.) Eighth, since each student can specify in the style panel parameters that fit his machine, students with machines on the lower-end do not lag behind the class. Finally, the hand-shaking approach which is used between Process 1 and Process 2 on the instructor's machine, comes to facilitate debugging, when the system is used by a single user.

8 Conclusion

We presented in this paper a conceptual model for an algorithm animation system for a constrained domain, that can be used in an electronic classroom. We also described a system for geometric computing that realizes this model. Geometric algorithms can be very complex, difficult to grasp, and hard to implement and debug. The visual nature of geometry makes animations extremely helpful. In such a well-defined domain it becomes practical to hide the graphics from the programmer and to automate the process of creating algorithm animations.

With GASP-II, programmers can produce animations in a matter of days, or even hours. The animations can be used to help explain algorithms, to check out ideas, and to debug the code. End–users can not only view the animation, but also interact with it, query about the algorithm, and collaborate with other users over the network. Though observing the same animation, each student can easily choose a different *style* of visualization. This is done without ever modifying or compiling the code.

In order to make algorithm animations a central part in teaching computational geometry, it is necessary to reduce the effort involved in creating animations. GASP-II is a step in this direction. Many possibilities exist in making an electronic book out of GASP-II. In this case, the user will be able to independently study a 3D geometric algorithm and experiment with its animation over the network. Finally, we hope that the conceptual model of GASP-II can be used for the creation of animation systems for other well-defined domains.

References

[1] J.E. Baker, I.F. Cruz, G. Liotta and R.Tamassia. Algorithm animation over the World Wide Web. *Proc. Int. Workshop on Advanced Visual Interfaces (AVI '96)*, pages 203-212, 1996.

[2] M.H. Brown. *Algorithm animation*. MIT Press, 1988.

[3] M.H. Brown. Zeus: a system for algorithm animation and multi-view editing. *Computer Graphics*, 18(3):177–186, May 1992.

[4] M.H. Brown and M.A. Najork. Collaborative active textbooks: a web-based algorithm animation system for an electronic classroom. *SRC Research Report 142*, May 1996.

[5] M.H. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2(1):28–39, Jan 1985.

[6] P. Epstein, J. Kavanagh, A. Knight, J. May, T. Nguyen, and J.-R. Sack. A workbench for computational geometry. *Algorithmica*, 11(4):404–428, April 1994.

[7] A. Lawrence, A. Badre, and J. Stasko. Empirically evaluating the use of animations to teach algorithms. *IEEE Symp. on Visual Languages*, pages 48–54, Oct. 1994.

[8] K. Mehlhorn and S. Näher. LEDA — a library of efficient data types and algorithms, *Lecture Notes in Computer Science 379* (Springer-Verlag, Berlin, 1989), pages 88–106.

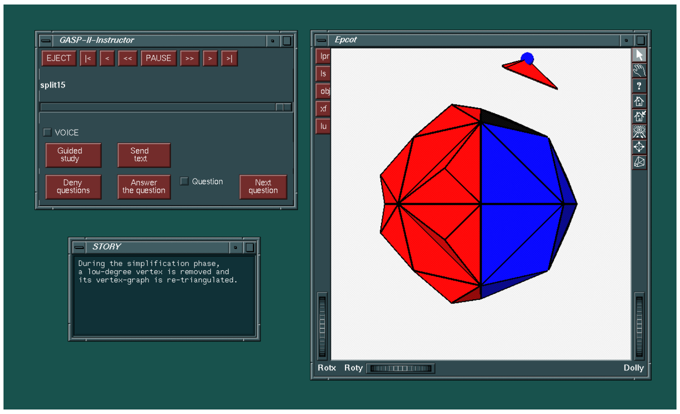
[9] P. Schorn. The XYZ GeoBench: a programming environment for geometric algorithms, *Lecture Notes in Computer Science 553* (Springer-Verlag, Berlin, 1991), pp. 187–202

[10] J. Stasko. Tango: a framework and system for algorithm animation. *IEEE Computer*, September 1990.

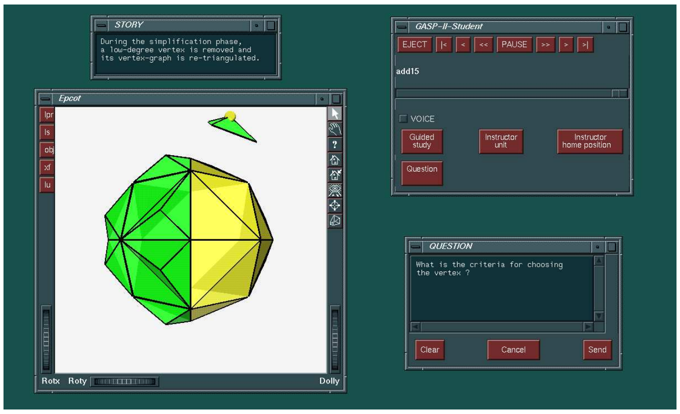
[11] A. Shapiro and A. Tal. Correspondence between 3D polyhedra for shape transformation. Technion – Israel Institute of Technology, EE PUB 1084, March 1997.

[12] M. Shneerson and A. Tal. GASP-II: a geometric algorithm animation system for an electronic classroom. The Weizmann Institute Technical Report, Department of Applied Mathematics and Computer Science, CS96-21.

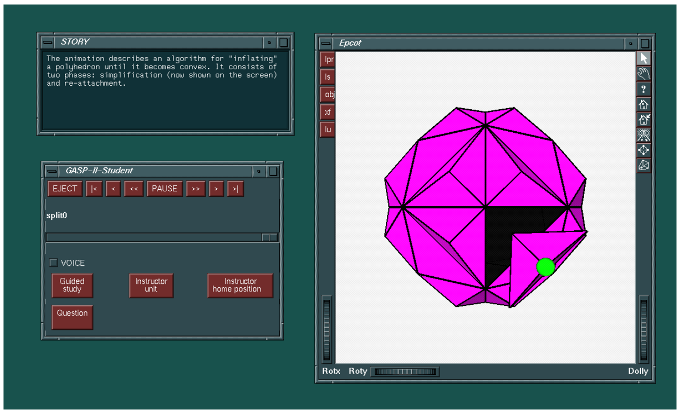
[13] A. Tal and D.P. Dobkin. Visualization of geometric algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 1:194–204, 1995.



(a) The Instructor's Screen

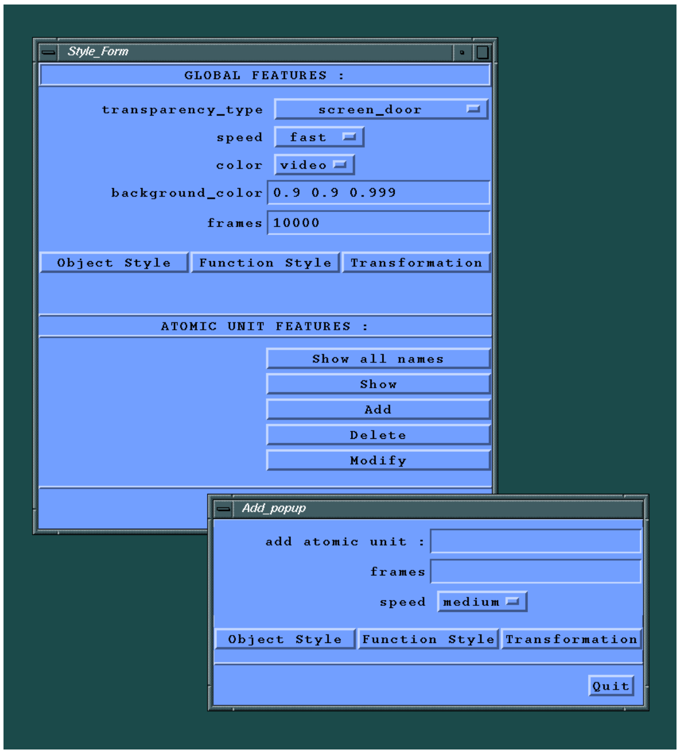


(b) A Student's Screen

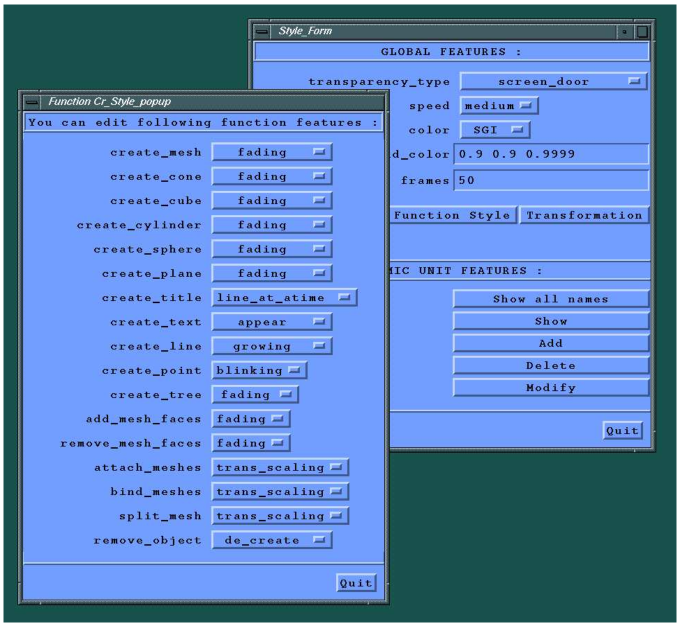


(c) Another Student's Screen

Plate 1: Snapshots of Screens



(a) Atomic Unit Style



(b) Function Style

Plate 2: Style Panel