Distributed Development and Teaching of Algorithmic Concepts

Seth Teller, Nathan D.T. Boyd, Brandon W. Porter, and Nicholas J. Tornow
MIT Computer Graphics Group
graphics.lcs.mit.edu

Abstract
We describe Fuse-N, a system for distributed, Web-based teaching of algorithmic concepts through experimentation, implementation, and automated test and verification. Fuse-N is accessible to, and usable by, anyone with a Java-enabled Web browser. The system was designed to:

1. Minimize the overhead required for students to engage in the essence of the learning experience.
2. Allow students to experiment with, generate, and evaluate algorithms and their implementations.
3. Facilitate greater, more effective interaction among students and between students and teaching staff.

The system represents algorithmic concepts as dynamic "modules" in Java. Students map inputs to outputs either via system-supplied "reference" implementations; manually, with corrective feedback; or by writing one or more alternative implementations for the concept. The output of the students' implementation is programmatically and visually compared to that of the reference implementation. Modules can be interconnected in a dynamic dataflow architecture of arbitrary complexity. Teaching staff can monitor student progress online, answer questions, execute student implementations, and perform a variety of administrative tasks.

A prototype system supported two modules. We have since extended Fuse-N to include a classical polygon rasterization pipeline, a module authoring "wizard," an editor, and several other components. We describe Fuse-N from student, staff, and developer perspectives. Next we describe its architecture and the technical issues inherent in its construction and extension. Finally, we report on some early experiences with the system.

Motivation
Computer science courses with large enrollments and significant programming components stretch the abilities of teaching staff to provide effective infrastructure, and individualized attention, to their students. For example, students need development tools and standardized (or at least consistent) environments in which to generate, test, and submit their implementations. Providing and maintaining these can be a formidable engineering task for the staff. Students submit complex programs requiring evaluation by the staff, often reducing the time available for individual attention to students. Also, despite nearly ubiquitous networks, email, etc., it is difficult, with standard mechanisms, to ensure effective, timely communication and collaboration among students and between students and staff.

The Fuse-N system addresses each of these considerations. First, it provides an intuitive, consistent visual interface through which students can experiment with, generate, and test algorithm implementations, ultimately submitting them for grading. This interface is backed by a robust client-server architecture for maintaining persistent versions of assignments and student implementations. Second, Fuse-N provides an interface through which the teaching staff can assess student work in progress and submitted work, using a location-independent mechanism for executing students' implementations. A development kit for generation of new modules is also provided. Third, the system provides mechanisms for synchronous and asynchronous communication among students and between students and staff, including annotations of course material, chat boards, shared whiteboards, and staff grading and feedback for specific student work.

Previous Work
The Web-based educational tool WebCT [Gol+96] provides sophisticated authoring tools to embed general course content, but no specific support for teaching algorithmic concepts is included. Instructional software development environments are less common, though the Jscheme environment [Hic97] allows students to implement assignments in a subset of the Scheme language from any Java-enabled Web browser. Algorithm visualization environments and toolkits, such as the Zeus system [Bro91] and Brown University's Interactive Illustrations [Dol97,Try97] provide sophisticated frameworks for examination of running algorithms. Finally, dataflow systems such as SGI's ConMan [Hae88], IBM's Data Explorer [IBM98], and AVS [Ups+89] allow direct manipulation and aggregation of functional modules. (An in-depth discussion of these and other related systems can be found elsewhere [Por98].) To our knowledge, however, Fuse-N is unique in its combination and synthesis of many such elements into a system for distributed collaborative pedagogy.

System Users' Perspective
One effective way to view the Fuse-N system is from a series of user perspectives. The system is first described from the point of view of a student. The teaching staff's point of view is described next, including mechanisms for disseminating course material, and collecting and evaluating student work.

Finally, the developer's perspective details the steps required to author a new module for the system. Our discussion largely avoids implementation details; these are deferred to Section 4 (System Designer's Perspective).

Student Perspective

Unorganized, the sheer volume of available course material can overwhelm a student. Thus the educator has two principal tasks: first, to select a collection of material to be taught, and second, to organize this material into a coherent progression of concepts. Fuse-N facilitates exactly this sort of progression in its exposure of concepts at several levels of abstraction (or, equivalently, at several levels of detail). For example, computer science and engineering courses typically emphasize the functional composition or interrelation of a collection of smaller pieces. Fuse-N represents such building blocks as algorithmic components to be manipulated by the student. These components are visible as modules within the Concept Graph, a gridded workspace containing a persistent collection of interconnectable modules selected by the course staff (Figure 1). Each module represents an algorithmic concept: a well-defined algorithmic mapping of input instances to outputs. Each module has a visual representation, consisting of an icon, a text label, multiple selectable input and output ports, and an interface with which the user can select among the interaction modes defined below.

In line with the usual notion of abstraction, students can think of modules and their connections either as closed functional units, or as open, modifiable components. For example, students can connect one module's output port to another module's input port (provided their types match), thereby assembling an aggregate functional unit. Students can open a module for editing, supplying their own code to implement the module's function. Successive student implementatinos can be compiled and reintegrated into the running system on-the-fly (Section 4.1).

Students who actively engage in implementing an algorithm understand and retain concepts better than those who simply, and passively, view visualizations [Law+94]. In Fuse-N, students can assess their understanding with the help of four interaction modes: Reference, Manual, Implement, and Difference. In all cases, algorithm inputs are either provided by the system or specified interactively by the student or staff. Regardless of mode, a diagnostic area reports a textual description of each input instance and each output action.

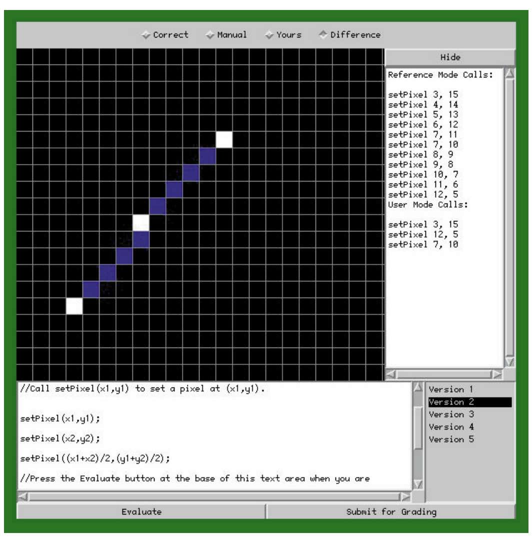

Figure 1 The Fuse-N work area

Reference Mode allows the student to view the operation of a correct "reference" implementation (cf. Figure 1) provided by the module developer (Section 3.3). The source code for the reference implementation is generally hidden from the student, though in some cases it may be exposed (for example when the student is challenged to implement some alternative, or asymptotically superior, strategy). Reference mode is similar to that provided by traditional algorithm animation and visualization systems [Bro+84]; it helps a student understand algorithm behavior for a variety of input instances, including boundary cases in which the correct behavior might not be otherwise evident.

Distributed Development and Teaching of Algorithmic Concepts

Manual mode challenges the student to effect the algorithm's operation solely through user-interface actions (Figure 2). Students need not do so by simulating an algorithm; their task is only to produce the correct output for the currently specified input. The system provides an immediate visual response to each student action. For instance, Manual mode for Bresenham's algorithm challenges the student to select, for a given line segment specified by its endpoints, those pixels which would be drawn by Bresenham's segment rasterization algorithm [Fol+82]. Correctly placed pixels are drawn normally, whereas incorrect pixels appear red. Manual mode inputs are typically crafted to allow students to generate the correct outputs in a few minutes.

Implement mode challenges the student to supply Java code that implements the algorithm for the current module (Figure 3). In this mode, the student is effectively presented with the interface to a Java class, whose methods are invoked by the system during interaction. Continuing with our example, Implement mode provides a stubbed method Bresenham(x1,y1,x2,y2), which is invoked whenever the student selects or specifies a point pair, or a suitable input event arrives from some upstream module. The interface also provides one or more functioning "base calls," which map to the module's abstract output ports. Thus the base call SetPixel(x,y) effects the illumination of the pixel at framebuffer position (x,y), causes the printing of a diagnostic message, generates a typed event for forwarding to downstream modules, or performs any combination of these, according to the current interaction context.

Difference mode displays a visual representation of the difference between the output of the student's implementation and that of the reference implementation (Figure 4). For example, difference mode for the Bresenham module colors pixels in one of four ways. Pixels drawn by both the reference and student implementation are shown in white; those omitted by the student algorithm are shown in blue; those drawn in the wrong position are shown in red; and those output more than once are shown in green. Difference mode is a particularly useful and unique feature of Fuse-N; it allows the student (and staff) to determine quickly the algorithm's correctness for a variety of prefabricated or interactively specified input instances. (Crafting efficient and pedagogically useful difference engines is a challenging part of developing a new module; we address this issue later.)

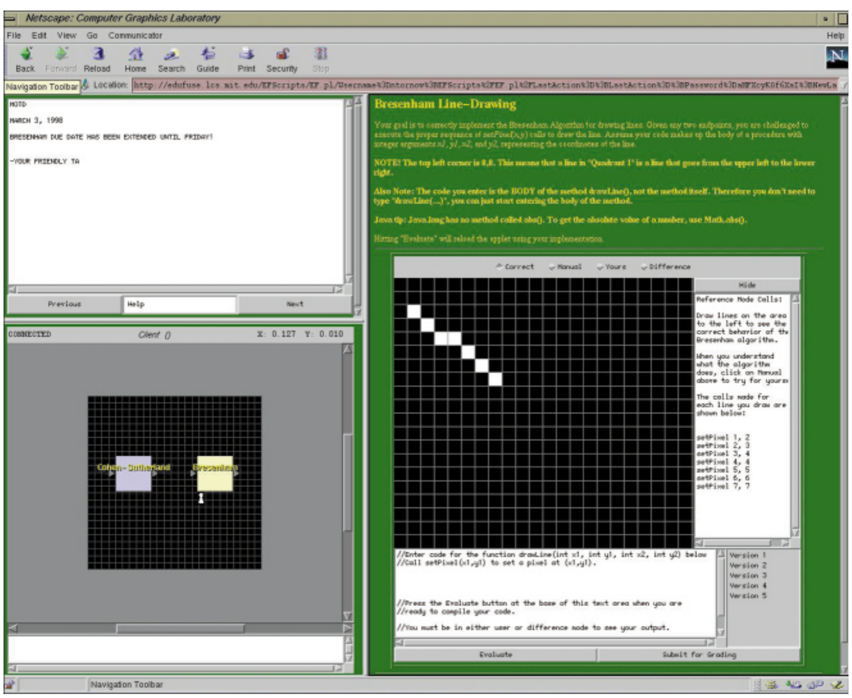

Figure 2 Manual mode

The student can invoke the Fuse-N editor for any module. This is a rudimentary editor that displays the class interface, staff comments and hints, and the student's code. It also allows the student to load, save, and compile implementations, and view compiler diagnostics. The editor will eventually incorporate debugging operations, student and staff annotations, and other administrative functions (see Current Efforts and Future Directions).

A suite of communication tools round out the student's view of Fuse-N. Students can read the message of the day, chat with other students or teaching staff who happen to be online, browse the chat history to look for questions that may already have been asked, or share a graphical whiteboard. We are extending these features to allow "sharing" of a client's workspace.

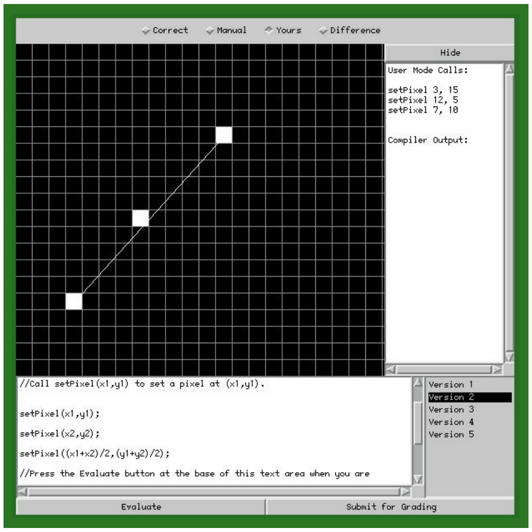

Figure 3 Implement mode

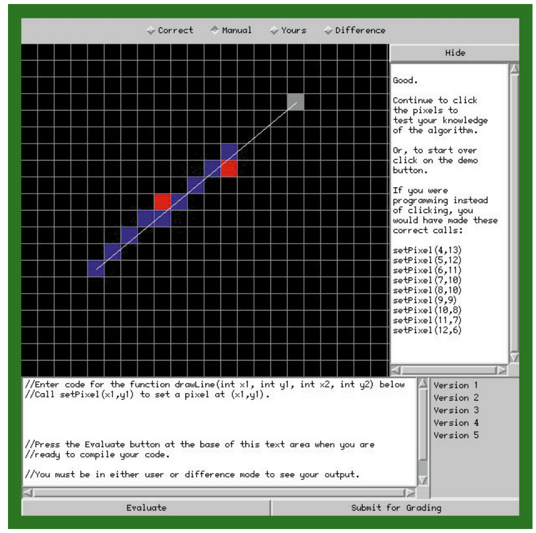

Figure 4 Difference mode

Thus students and staff will be able to interact with a Concept Graph and modules as if they were at the same console; for example, a teaching assistant could demonstrate a certain input case to the student by direct manipulation of the student's desktop. This and many other collaboration aspects of Fuse-N are discussed elsewhere [Boy97].

Teaching Staff's Perspective

Fuse-N is designed to maximize the teaching staff's availability and effectiveness through provision of infrastructure for creation, selection, distribution of course material, tracking and evaluation of student work, and general course administration such as maintenance of course lists, lecture notes, grades, daily messages, and other materials.

Generating, testing, organizing and distributing course software infrastructure to students can be a significant burden for teaching staff. Fuse-N provides a simple mechanism for authoring modules that represent algorithmic concepts, and functionally interconnecting these modules within a dataflow architecture with visible structure. Related materials are available through standard Web mechanisms like linking.

One traditional coursework model involves the staff's generation and assignment of work: assignment completion; a deadline at which time students turn in completed work; an evaluation period for staff to process this work; and a feedback period in which a grade and other information are returned to the student. These phases routinely span several weeks of activity.

Fuse-N provides several mechanisms that extend this batch model to an interactive setting. For example, the system places a visual avatar for each student near the visual icon representing the module on which the student is working (Figure 2). The teaching staff can broadcast timely messages to selected groups. The staff can also monitor an individual student's progress by locally executing the student's current implementation in Implement or Difference mode as described above. An assistance queue, interactive chat sessions, whiteboards, and this "over-the-shoulder" execution mode all increase the staff's ability to identify struggling students and help them in a timely fashion.

Several mechanisms are provided to ease the process of evaluating large amounts of student code. The staff can apply prefabricated test cases to student implementations. Common errors are then identified by interposing the student module between a generate/test module pair. A Web-based grading form (Figure 5) prompts the staff member for specific feedback about the student work and can incorporate input instances that cause the student implementation to exhibit specific behavior. These mechanisms log grading information and associated material to the server for persistent storage.

Finally, the system provides a uniform, location-independent interface to all of these mechanisms. Thus just as the students may complete coursework from any browser, so can the staff grade or otherwise provide feedback to any student from any browser (Figure 5). We have found this capability to be of significant value in actual course administration (Section 5.1).

Creating a new module is a relatively simple process. Using common design patterns, the basic functionality of every module can be coded automatically. Through a dialog box, the developer specifies the module name, the input type(s) that the module can receive, and the output type(s) that the module generates. A Java servlet, analogous to a development kit "wizard," generates a Java source file representing the module and compiles it into a class file. The Fuse-N client then requests that the class be loaded into the current application, creating an editable instance of the new module.

invoking a provided method call. The developer must also design, specify, and provide the base call(s) with which the student is to generate output. Each base call must then be mapped to the appropriate output port. Finally, the developer must specify and implement the module's behavior in Manual, Reference, and Difference modes. Once a module has been created, it is easily added to the Fuse-N server, becoming available to all clients of that server.

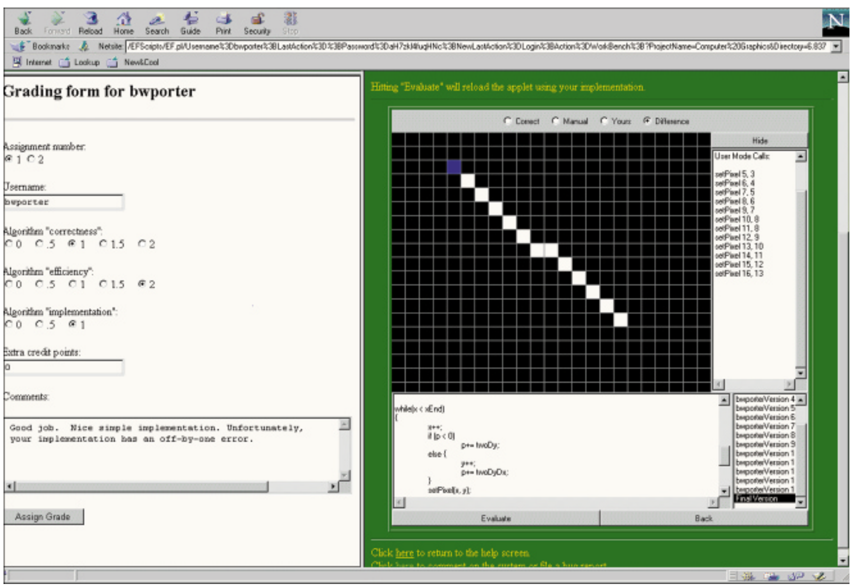

Figure 5 Grading a student from within a browser

Module Developer's Perspective

This section describes the system from the point of view of the module developer, who maps abstract algorithmic concepts to actual modules in Fuse-N. Module developers may be teaching staff, students, or more broadly any third party who wishes to generate course content.

In order to implement a specific module, the module developer first provides code that maps both input and output data to visual representations on a graphics canvas associated with the module. Next, the developer codes the actions to be taken when new data arrive on any input port. The developer issues data on an output portsimply by

System Designer's Perspective

This section describes Fuse-N from the point of view of its designers. Here we discuss the technical underpinnings of the system and some of the issues inherent in its construction and extension.

System Architecture

This section briefly outlines the technologies used in the system architecture. Specifically we discuss our choice of a client-server model, the advantages of location- and platform-independence, and Fuse-N's dataflow architecture.

Fuse-N is implemented as a client-server architecture. The Fuse-N server is comprised of a traditional Web server and a group of Java servers and Java servlets. A Java registry server tracks users in the system; a Java messaging server handles text-based message delivery; and a Java servlet performs essential system functions such as compilation of user modules and generation of HTML code for display to the client. All user, course, and system information is stored persistently on the server. The server supports location-independent access to user data, restricting access with a simple password mechanism.

The client application is comprised of an HTML environment and a set of Java applets, both of which are served to the user's browser. Users receive the most recent version of the Fuse-N client at the start of each session. Because Fuse-N is coded entirely in Java, Fuse-N clients and servers can run on any platform equipped with a suitable Java Runtime Environment [Gos95].

Fuse-N uses a stateful dataflow model of sequential execution, enabling arbitrary aggregation of components. Software applications can be created by linking the output(s) of stateful components to the input(s) of other components. Thus an application can be modeled as a directed acyclic graph of components. Because the input(s) and output(s) of the components in a dataflow system are defined explicitly, each component can be treated as a black box abstraction of the operation it performs. The resulting dataflow aggregations are modular and can be rapidly assembled, modified, and extended. Dataflow systems naturally extend to parallel operation, as each component processes data independently of all other components.

Fuse-N's underlying architecture supports the ability to load modules, manage interconnections, and route information from any output to any input. Thus users create complex systems by linking modules in a run-time dataflow system, as in ConMan [Hae88]. Individual components have graphical representations that can be arranged and linked by the student. Links are typed and can be created, modified, or destroyed.

Modules run in separate threads. Each module queues incoming data for processing and outgoing data for delivery to downstream modules. Modules can be recompiled dynamically; successfully compiled modules are inserted into the running dataflow. Newly created modules inherit all of these capabilities through a subclassing mechanism. The student is left to focus entirely on the module algorithm, interconnections, and graphical display. (A comprehensive overview of Fuse-N's dataflow architecture can be found elsewhere [Bwp98].)

Early Experiences

Two specific experiences with Fuse-N have been illustrative. First, we set out to assess the system's effectiveness by deploying two simple modules as assignments for an undergraduate computer graphics class at MIT. Second, we implemented a "classical" polygon rasterization pipeline as a series of dataflow components within Fuse-N. After reviewing these experiences, we describe several current and future directions for Fuse-N.

A Trial Run

Fuse-N's early prototypes were designed primarily as proofs of concept. Fuse-N was first deployed to students in the fall of 1997, in the form of an introductory computer graphics course with 104 students. Each student was asked to implement two modules inside Fuse-N: Bresenham's segment-drawing algorithm and Cohen-Sutherland's segment clipping algorithm [Fol+82]. Upon submission of the assignment, each student was asked to fill out an anonymous survey. The results were positive; students felt the assignments were well organized and helped them learn the material.

As expected, human interaction was a crucial element of the process. The collaboration tools of Fuse-N were heavily used, allowing teaching assistants to pinpoint many of the errors students were making only minutes after students began coding. With the students' permission, teaching assistants effectively provided preemptive assistance. They did not have to wait for students to start asking questions, but could load students' code and examine its behavior. The staff could then clarify technical issues and make suggestions to the entire class, or only to those students working on a specific module. This immediate problem detection and correction ability allowed students and teachers to interact effectively outside of the classroom.

Another interesting aspect of the system was the staff's ability, given access to the development process as well as the final results, to distinguish among the learning and implementation styles of the students. Some students wrote careful pseudocode, then a small number of implementations. Others seemed to be coding by successive approximation, submitting many scores of slightly varying code revisions until a correct implementation was finally achieved. This rather unexpected result - that in removing essentially every barrier to recompilation we may have actually encouraged an inferior programming practice - has led us to reexamine the way in which students are asked to use the system. In particular, we are now investigating mechanisms by which students can be urged to achieve greater mastery of the material before attempting implementation. For example, students could be prompted to demonstrate understanding of the input/output mapping in Manual mode, and to submit coherent pseudocode to a staff member, before commencing implementation.

Implementing a Polygon Rasterization Pipeline

We extended, and successfully tested, the system by implementing a classical polygon rasterization pipeline from a core set of modules (Figure 6) including TraverseScene, CameraModel, BackFaceWorld, WorldToEyeSpace, EyeSpaceToNDC, ClipNDC, NDCToScreenSpace, RasterizeTriangle, DepthBuffer, and FrameBuffer. Each member of the development team implemented several modules. Instantiating each module required clicking the New Module button, entering the module's name, and specifying the module's typed inputs and outputs. A compilable stub for the module was then automatically generated and compiled by Fuse-N. Finally, we filled in the stubs by writing Java code to effect the functional mapping required of each module and wired together the appropriate module inputs and outputs through the Concept Graph interface. Implementing this system took three students about one week. This was less time than we had anticipated. (However, the students implemented only the Reference algorithm for each module, arguably the easiest portion; the Manual and Difference mode implementations are underway.) The system proved robust, supporting multiple simultaneous execution paths while maintaining responsiveness. The dataflow model was a natural fit to the graphics pipeline. Students who implemented parts of the pipeline reported that even though they previously understood the pipeline, they had a much better sense of it after implementing it in Fuse-N. The students also learned that adding internal, "self-checking" routines to modules was a particularly useful technique. Overall, the Fuse-N environmentmade the process of building and exercising this system efficient and educational.

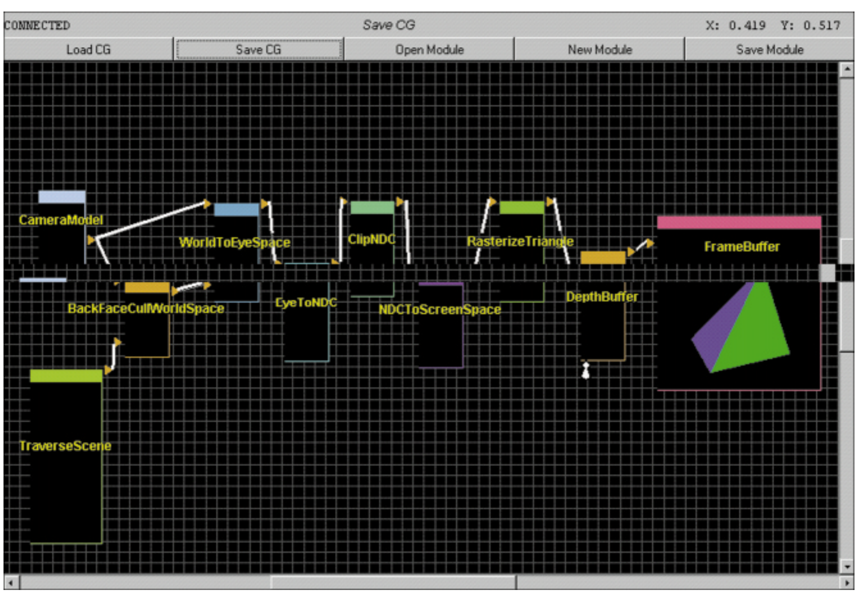

Figure 6 A Dataflow Polygon Rasterization Pipeline

Current Efforts and Future Directions

We continue to add capabilities to the system. Our active research and development efforts can be categorized as: improving the integrated environment for students and developers; crafting new capabilities for the teaching staff; and extending the dataflow model.

The current editing and debugging capabilities are adequate for simple modules but ill-suited for more complex implementation tasks. We are actively incorporating an improved code editor and debugger into the system. Mechanisms for more closely coupling applets, dynamic module state, and compiler and run-time diagnostics are being explored as well.

We are also expanding the capabilities of the teaching staff by improving the system's ability to track students. More generally, we are enhancing Fuse-N's general administrative features. For example, the staff expressed the desire to revise grading criteria, generate Web-based grading forms, and analyze and distribute grades from within the system. Finally, we have designed for transparent sharing of events across the network [Boy97]. This will allow teaching assistants to effectively reach over a student's shoulder and assume control of the mouse or keyboard, providing a kind of remote tutoring.

A related thrust of our work is extending modules to connect to other modules on remote machines, using a transparent network event layer [Boy97]. Providing basic dataflow constructs will help students and teaching staff assemble complex systems easily. Abstracting these connective elements as components in their own right is a notable innovation of our system. Finally, determining how best to integrate manual interaction with the dataflow system and accompanying visualizations, and how to simplify generation of difference engines for complex algorithms, are both hard open questions.

We foresee developing a number of additional modules as part of the Fuse-N project to augment the full undergraduate computer graphics curriculum currently under way. We are exploring collaborations with other algorithmic learning domains, such as those of MIT's class 6.001, The Structure and Interpretation of Computer Programs [Abe+85]. There, we plan to support a Scheme interpreter within a Fuse-N module, allowing dataflow interconnection of "Schemelets." We hope that as Fuse-N becomes more widely used, students, educators and researchers will contribute modules to the system, creating a distributed library of instructional algorithmic modules in the public domain.

Conclusion

Fuse-N improves upon traditional software development environments, while extending Web-based educational systems to support experimental problem solving. We designed it explicitly to support interactive pedagogy of algorithmic concepts. The system supports rapid engagement of students with the essence of the material. Fuse-N supports algorithmic instruction through Reference, Manual, Implement, and Difference modes for each module. At a higher level of abstraction, students can link modules in a dataflow environment. Preliminary results are encouraging, as gauged both by classroom experiences with an early prototype and by our nascent realization of a classical polygon rasterization pipeline within Fuse-N.

Acknowledgments

We gratefully acknowledge the past and present Fuse-N development team, including the authors: Bhuvana Kulkarni, Aaron Boyd, Randy Graebner, and Arjuna Wijeyekoon. The Fuse-N project has been supported by the MIT Class of 1951, the MIT Lab for Computer Science, the National Science Foundation (through Career Award IRI-9501937), Silicon Graphics, Inc., and Intel Corporation.

References/Bibliography

[Abe+85] Abelson, Harold and Sussman, Gerald and Sussman, Julie. Structure and Interpretation of Computer Programs. MIT Press/McGraw-Hill, 1985.

[Boy97] Boyd, Nathan. A Platform for Distributed Learning and Teaching of Algorithmic Concepts. MIT MEng Thesis, 1997.

[Bro+84] Brown, Marc and Sedgewick, Robert. A System for Algorithm Animation. SIGGRAPH Proceedings: Volume 18, Number 3. July 1984. Pages 177 - 186.

[Bro91] Brown, Marc. Zeus: A System for Algorithm Animation and Multi-View Editing. In IEEE Workshop on Visual Languages, pages 4-9, October 1991.

[Dol97] Dollins, Steven. Interactive Illustrations. http://www.cs.brown.edu/research/graphics/research/illus/. Brown University Department of Computer Science, 1997.

[Fol+82] Foley, James, and van Dam, Andries, Fundamentals of Interactive Computer Graphics. Addison-Wesley, 1982.

[Gol+96] Goldberg, Murray et al. WebCT - World Wide Web Course Tools. http://homebrew.cs.ubc.ca/webct/. University of British Columbia Department of Computer Science, 1996.

[Gos+95] Gosling, James and McGilton, Henry. The Java Language Environment: A White Paper. Sun Microsystems. October 1995.

[Hae88] Haeberli, Paul. ConMan: A Visual Programming Language for Interactive Graphics. SIGGRAPH Proceedings: July 1988. Pages 103-111.

[Hic97] Hickley, Tim. Jscheme: an Applet for Teaching Programming Concepts to Non-Majors. http://www.cs.brandeis.edu/~tim/Packages/Jscheme/Papers/jscheme.html Brandeis University, Michtom School of Computer Science, 1997.

[IBM98] International Business Machines, Inc. IBM Data Explorer (DX), http://www.almaden.ibm.com/dx/, 1998.

[Law+94] Lawrence, Andrea, and Stasko, John, and Kraemer, Eileen. Empirically Evaluating the Use of Animations to Teach Algorithms ftp://ftp.cc.gatech.edu/pub/gvu/tech-reports/94-07.ps.Z Technical Report

GIT-GVU-94-07. Georgia Institute of Technology College of Computer Science. 1994.

[McC95] McCanne, Steve, and Jacobson, Van. vic: A Flexible Framework for Packet Video, ACM Multimedia '95.

[Por98] Porter, Brandon. Educational Fusion: An Instructional, Web-based, Software Development Platform. MIT MEng Thesis, 1998.

[Try97] Trychin, Samuel. Interactive Illustration Design. Brown University Computer Graphics Lab, 1997.

[Ups+89] Craig Upson, Thomas A. Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: a Computational Environment for Scientific Visualization. IEEE Computer Graphics and Applications, Vol. 9, Num. 4. p.30-42. July 1989.