GeoBuilder: A Geometric Algorithm Visualization and Debugging System for 2D and 3D Geometric Computing

Jyh-Da Wei, Ming-Hung Tsai, Gen-Cher Lee, Jeng-Hung Huang, and D. T. Lee, *Fellow, IEEE*

***Abstract*— Algorithm visualization is a unique research topic that integrates engineering skills such as computer graphics, system programming, database management, computer networks, etc., to facilitate algorithmic researchers in testing their ideas, demonstrating new findings, and teaching algorithm design in the classroom. Within the broad applications of algorithm visualization, there still remain performance issues that deserve further research, e.g., system portability, collaboration capability, and animation effect in 3D environments. Using modern technologies of Java programming, we develop an algorithm visualization and debugging system, dubbed *GeoBuilder*, for geometric computing. The GeoBuilder system features Java's promising portability, engagement of collaboration in algorithm development, and automatic camera positioning for tracking 3D geometric objects. In this paper, we describe the design of the GeoBuilder system and demonstrate its applications.**

***Index Terms*— Computational Geometry, Geometric Algorithm Visualization, Knowledge Portal, Collaborative Design, Camera Position, LEDA, Convex Hull, Line Segment Intersection**

I. Introduction

ALGORITHM animation and visualization techniques are used to graphically represent program states during the execution of a particular algorithm. Development and evaluation of visualization tools have become an active research topic of computer science not only because visualization is emphatically used to help humans understand scientific progress, but also because computers are now in widespread use and are quite convenient for creating visualizations [11]. Recent research of algorithm visualization utilizes engineering skills such as computer graphics, system programming, database management, computer networks, and human-computer interface etc., to build an overall system for academic research and computer science education[1] [18], [29]. In this paper, we present a geometric algorithm visualization and debugging system, *GeoBuilder*, for 2D and 3D geometric computing. This system can facilitate geometric algorithmic researchers in not only testing their ideas and demonstrating their findings, but also teaching algorithm design in the classroom. We have embedded GeoBuilder into our previously developed OpenCPS (Open Computational Problem Solving) knowledge portal [25], [51] as a practice platform for lecturing on a graduate class, geometric computing and algorithm visualization.

The GeoBuilder system possesses three important features making it more promising than any other existing visualization system. First, GeoBuilder is a platform-independent software system based on Java's promise of portability, and can be invoked by Sun's Java Web Start technology in any browser-enabled environment. Second, GeoBuilder has the collaboration capability for multiple users to concurrently develop programs, manipulate geometric objects and control the camera. Finally, the 3D geometric drawing bean of this system provides an optional function that can automatically position the camera to track 3D objects during algorithm visualization.

The effectiveness of our system can be observed by using Naps et al.'s criteria of visualization [21], [31]. Naps et al. suggest six levels of learner engagement in a visualization tool, i.e., no viewing, viewing, responding, changing, constructing, and presenting. This taxonomy lists engagement levels that benefit learners increasingly from passive levels to the most active. With the exception of non-viewing level, viewing is the most basic requirement of a visualization system. Beyond that, the responding level supports question-and-answer activities asking users to anticipate the next frame and/or to indicate the program segment of a running algorithm; then, the changing level allows users to modify variables and explore the algorithm's behavior. Many visualization tools have reached the fifth level – PDE.Mart [29], PETSc [20], [34] and DEAL.II [3], [2] are examples that allow users to construct their own visualizations of the algorithm under consideration. Meanwhile, the highest, i.e., presenting, level requires users to explain the visualized algorithm to an audience. It is not easy to seamlessly integrate the function into a visualization system.

Our system reaches the uppermost level because of its portability, concurrent collaboration capability and runtime visualization utility. Java's portability allows the GeoBuilder system to be plugged into the OpenCPS knowledge portal as a standard implementation interface. Following the knowledge management policy, students can present their perspectives about the implementation codes. On the other hand, the features provided, such as collaboration facilities during program development, and utilities for manipulating geometric objects for visualization, allow users to demonstrate their ideas, and to achieve on-line presenting effects.

Aside from system portability and collaboration capability,

Authors are with the Institute of Information Science, Academia Sinica, Taipei, Taiwan. E-mail:{jdwei, mhtsai, gc, jhh, dtlee}.iis.sinica.edu.tw. G. C. Lee and D. T. Lee are also with the Department of Computer Science and Information Engineering, National Taiwan University.

[1]Sometimes, visualization systems are referred to as problem-solving environments by their authors for they fully support visualization components, numerical simulations, and other computational facilities required to solve a target class of problems [10], [28], [29].

we do take into account the animation effect – the kernel issue of visualization systems. The most interesting design for the GeoBuilder system is the dynamic decision of camera position for 3D algorithm visualization. Camera position is not a problem in 2D environments[2]. However, it becomes serious in 3D environments because the geometric objects being visualized may overlap in front of the camera view. The 3D drawing bean of GeoBuilder has an extensible rule-base allowing the camera to move to a proper position to focus on the objects in the next step, and we have suggested a simple decision rule that will be described later. In the following section, we introduce the related literature of this work. Then we explain the system architecture, the implementation of collaboration, and the 3D drawing bean with automatic camera positioning in Sections III, IV and V respectively. Finally, future work and conclusion are given in Section VI.

II. Background and Related Work

Brown and Sedgewick [8] published the BALSA (Brown University Algorithm Simulator and Animator) system as the first fully-functioning algorithm animation system in 1984. Development of algorithm visualization systems has received increasing attention ever since. In 2004, Hendrix [16] has categorized algorithm visualization systems into three classes: "general purpose software and algorithm animation systems", "software and algorithm animation systems for data structures" and "software and algorithm animation systems for debugging." The first class requires the users to create visualizations by inserting specific program segments and function calls directly into the source code in the animation systems. The second class centers on data structure visualization and provides users with a rich environment to interact with data structure. Finally, the third class bridges the gap between debugger systems and sophisticated algorithm animation systems. Our system provides a visual debugging environment with full support of the LEDA [19] library for geometric algorithm development and thus belongs to the last class.

In Subsection II-A, we survey the 3D algorithm visualization systems in this class. Most of these systems were built using the C language and thus they should be installed before executing. The state-of-the-art Java-based system JCAT [30] was developed in 2001 when the modern JOGL (Java bindings for OpenGL) techniques were not yet released. JCAT can run as a Java applet inside a web page; however, its drawing engine is built upon outmoded *Java 3D* libraries and does not support smooth camera moving and zooming modes. Our work focuses on 3D geometric algorithm visualization; therefore, we do not go through other worthwhile topics such as data structure visualization and model-view mapping as described in jGRASP [16], JIVE [9] and Vizz3D [35] projects.

The proposed GeoBuilder system and the Geo3D-DrawingBean thereof are portable software on the basis of the Java language and the JOGL bindings. A practical application of this system is to plug into the OpenCPS knowledge portal as an implementation interface. Therefore, we introduce the OpenCPS in Subsection II-B. Prior similar systems enabling collaborative development are then compared in Subsection II-C, followed by a description of the dynamic camera positioning feature in the last subsection.

[2]We are not saying that 2D camera control is trivial. This work only considers a static viewport for 2D visualization because this viewport is practical for normal input instances that do not exceed the bounding box of the scene. In extreme cases, the input objects may be too many and too large for the static viewport to present. However, these cases are likely unnecessary to discuss in algorithm development and demonstration.

A. Three Dimensional Algorithm Visualization Systems

Three-dimensional algorithm visualization systems can be categorized into C/C++ and Java-based families. It is quite natural that the systems using C/C++ were developed much earlier, since the C language has a longer history. For example, Polka-3D [42] and Zeus-3D [7] were first developed in 1993. Polka-3D seems to be the leader which suggested the object-oriented animation methodology. This system provides reusable classes to model animation effects, 3D objects, and special viewers etc. Polka-3D is GL (industrial Graphics Library) compliant, but was implemented only to run on the Silicon Graphics IRIS (SGI) workstation. Zeus-3D is an algorithm visualization system which does not purpose to show intrinsically three-dimensional objects. This system uses 3D concepts and graphical display to enhance the quality when presenting information of 3D objects contained in an algorithm.

In 1995, GASP [44] implemented a library of primitive animation functions. The input and output of this system are transmitted by files. GASP only runs on the SGI workstation with an exclusive UNIX operating system, which allows GASP to set its user interface (UI) upon Inventor and Motif/Xt. In the same year, Geomview [32] was released with a revolutionarily useful viewer for visualizing objects in three and higher dimensions. It reads geometric data from a simple formatted file and displays the objects in a window. Users can translate, rotate and scale the objects with mouse operations. However, Geomview is still built on the C language and can only run on a specific Unix platform.

Following the architecture of GASP, GASPII [40] was released in 1998. This upgrade maintains several screens when users run an algorithm. One is the instructor's screen while the other is the student's. Design like this helps in educational activities. GASPII runs in the same environment as GASP. Next in 1999, DAViD [26] proposed the 3Dsheet [24] framework to achieve 3D visualization capability. Using a mouse driven drag-and-drop mechanism, it developed the 3D-snap environment to facilitate computer-assisted design.

GeoWin [6] was the state-of-the-art system released in 2002. This system can be easily interfaced with renowned standard geometric software libraries including LEDA and CGAL. For instance, it has access to a simple 3D viewer based on the LEDA d3 window class. GeoWin owns input and output scenes. Every editing operation of the input interactive interface maps to an associated event. Whenever the input scene is modified, the output scene is recomputed and redrawn immediately.

Java-based algorithm visualizations systems are relatively fewer compared to those written in C/C++, mainly due to the

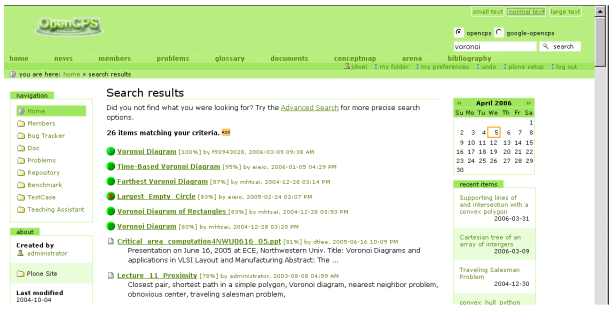
fact that Java is a newer language invented only in 1995. The 3D graphic interfaces of Java, such as Java 3D, were developed much later in 1999. Therefore, we can only find GAWAIN [15] and JCAT [30] released in 1998 and 2001 respectively. They both are visualization systems running as Java applets. Each user can interact with the 3D views locally, independent of the page, to animate geometric algorithms. The JCAT system further allows the user to start and stop the algorithm, advance the algorithm step by step, and adjust the visualization speed. In this paper, we will present the GeoBuilder, which was developed as a new generation of algorithm visualization and debugging system for geometric computing. Benefiting from modern JOGL technologies, this system is not only portable in web-based environment, but also equipped with camera manipulation capabilities.

B. OpenCPS Computational Problem Solving Environment

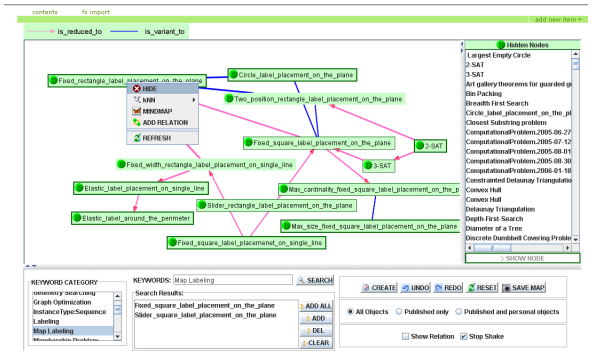
In previous work, we implemented a knowledge portal for Computational Problem Solving [25], dubbed *OpenCPS*, which is intended to combine geographically distributed efforts in algorithmic research. Algorithmic researchers are invited to share their information in the form of files, documents, etc., and collaboratively design or implement new algorithms. Knowledge in OpenCPS, consisting of various objects and content types, is regarded as a map relating instances in problem, solution, and implementation spaces. Problem space objects consist of uniquely identifiable computational problems, solution space objects consist of algorithmic solutions, and implementation space objects consist of actual code for the solutions.

All these knowledge objects have their own attributes. A computational problem possesses attributes such as problem name, description, problem category, equivalent problems, sub-problems, super-problems, variant problems, formal definition, input variables, output variables, problem status, existing solutions, and related publications. An algorithmic solution is associated with the following attributes: solution name, target problem, description, pseudo code, complexity, existing implementations, and related publications. As for the implementation space, the associated attributes are implementation name, target solution, description, related publications, as well as a folder of source code, running environment, and a launch button. Additional details can be found in [25]. The GeoBuilder system is a collaborative visual debugging system for geometric computing. It can be ported as a stand-alone application for users to develop, deposit, and demonstrate the algorithm codes collaboratively. In this work, we embed the GeoBuilder system as a plug-in on the OpenCPS website. Users can designate GeoBuilder as the running environment in the implementation space and then invoke it in the web environment directly. The effect of visualization engagement can therefore be improved to reach the highest, i.e., the presentation, level for computer assisted learning.

Fig. 1(a) shows the screenshot of the OpenCPS portal with search results of Voronoi diagram. Fig. 1(b) is a map generator we developed to observe group knowledge grown within the OpenCPS portal. A major application of the knowledge



(a)



(b)

Fig. 1. Screenshots of the OpenCPS portal. (a) Knowledge objects as the searching results of Voronoi diagram; (b) Map generator to observe the formation of group knowledge.

management (KM) is to support learning activity with the ability to organize information and knowledge resources on the web [1]. The contents of the OpenCPS knowledge portal are contributed by all its members, and thus can be regarded as group knowledge. Interestingly, these knowledge objects are not only represented in web pages but also associated with conceptual elements defined within the ontology of algorithmic problem-solving knowledge. In this regard, we developed a Site-And-Concept (SAC) map generator to visualize the formation of group knowledge [27]. This map generator, as shown in Fig 1(b), supports both expert and learner modes. Users can contribute to the group knowledge as well as learn from it through the SAC map.

C. Concurrent Collaboration Mechanism

The GeoBuilder system proposed in this paper supports algorithm visualization and debugging for geometric computing. Furthermore, we designed this system to support online collaboration among multiple users so that it enables a collaborative-learning and interactive-demonstration environment. Because of increasing availability of computer networks and the trend towards teamwork, collaboration enabling groupware has received much attention in the research field of CSCW (Computer Supported Collaborative Work) systems. We distinguish our system from other prior work with an implementation of concurrent collaboration. To clearly elucidate our work, we categorize the online collaboration systems into three levels: *desktop sharing*, *floor transfer*, and *concurrent control*.

As an elementary class of online collaboration, desktop sharing systems allow multiple users to work on a single

virtual desktop through their local input devices such as keyboards and mouses. Although the users can conveniently connect their terminal devices to a remote desktop, concurrent operation may cause mutual conflicts due to numerous and disorderly hardware interrupts made by different users at the same time. In the extreme case, users can neither move their mouse nor type a word. Microsoft VNC and NetMeeting systems and other commercial products like PC Anywhere are examples of this class. The mutual conflict problem can be solved by a floor transfer mechanism, which is adopted by the second class of online collaboration systems such as Ensemble, GroupDraw, and GroupGraphics [36]. Using a conflict prevention approach based on locking [12], these systems require the users to place a lock on a target object before they can access it. Conflicting operations are therefore prevented; meanwhile, the effectiveness of collaboration is decreased because the systems allow only one user at a time to operate the objects. In case the user forgets to release the floor, i.e., unlock the objects, the collaborative work will be blocked for a long while until the coordinating process detects the fault and resolves it.

We consider the highest level of online collaboration to be one that incorporates an implementation of concurrent control. This type of collaboration allows users to operate their own terminal devices concurrently with mutual awareness and consistency maintenance. In doing this, a session management server is required to coordinate the groups of replicated applications. Roseman and Greenberg initiated this research field by publishing the first version of GroupKit/ GroupSketch in 1992 [37], [38]. This system is built upon the C++ language and the INTERVIEWS toolkit of X-Windows. Although the INTERVIEWS toolkit has a limitation on creating arbitrary object classes, the idea of using remote procedure calls (RPCs) to communicate, share information, and trigger program execution between replicated application processes proved successful and has been extensively applied by its successors. Among the follow-up studies, the GRACE (graphics collaborative editing) system released in 2002 [43] stands out as the authors, Sun and Chen, proposed a distributed algorithm that ensures causality preservation for the underlying RPC operations. The algorithm makes it easy to create geometric objects in the GRACE system with causality preservation of user operations.

In comparison with the RPC type approaches, IBM's Jazz project [17] suggested a quick-upgrade system architecture reusing floor control techniques to realize concurrent collaboration. This project has demonstrated a system that embeds collaborative features such as chat, whiteboard, data sharing, and collaborative editing (co-editing) functions within the Eclipse application development environment. Importantly, the co-editing function thereof is implemented following an *into context* concept, rather than using the tightly coupled RPC approaches. This alternative method, referred to as *contextual collaboration* [14], settles co-editors in exclusively locked blocks and thus protects their work from mutual influence. The Jazz system is efficient in collaborative text editing; however, the contextual collaboration approach is not easy to extend to a graphic editing environment because it remains difficult to assign co-editing users to exclusive editing blocks.

Aside from RPC type approaches and contextual collaboration, a native feature of the Java language implies other possibilities to develop concurrent collaboration systems. As we know, an event is represented as an object in Java. When an event occurs, the Java system creates an event object and sends it to the registered listener objects. Following this event-driven methodology, we developed the *Collabench* (a shorthand for *collaborative workbench*) package in our ShareTone project [53]. Wrapping Local Objects (LocalObject) with Collaborative Objects (CollabObject), the Collabench maps Local Events (LocalEvent) to Collaborative Events (CollabEvent) and broadcasts these events for concurrent action. This package builds an event driven collaboration framework offering adequate classes and interfaces that are more portable than RPC type approaches and can facilitate the development of collaborative application. In this paper, we use Collabench to implement concurrent collaboration, since it is also suitable for collaborative graphic editing and thus more useful to geometric computing.

D. Camera Positioning in 3D Environments

Dynamic camera positioning has received increasing attention in the fields of computer vision [13] and virtual reality [5], [46]. A typical paradigm of computer vision is to determine the set of objects visible from a given viewing area and derive a camera placement method for generating image-based models [13]. Virtual reality technologies connect the image-based models to render a cinematic style, and thus support virtual 3D simulation, training, and entertainment environments with intelligent visualization interfaces [5], [46].

The GeoBuilder system implements the concept of camera positioning for 3D geometric algorithm visualization. Camera position is not a problem in 2D environments. This issue becomes serious in 3D environments, however, because the visualized geometric objects may overlap in front of the camera view. Therefore, we need a decision rule to move the camera to a proper position to focus on the objects in the next step. In Section IV, we will present the positioning rule based on an observation sphere. The camera will be moved along the surface of this sphere, located on one of the candidate positions, and then focused on the center of next object of interest.

III. System Architecture

We create the GeoBuilder system as an Integrated Development Environment (IDE) that supports LEDA [19] library for geometric algorithm development and visualization. Using this system, users can edit, debug, and visualize geometric algorithms written in C/C++. GeoBuilder provides a widget to generate frame code with geometric object classes. As Table I lists, the currently provided objects include points, line segments, polygons, etc., and lists of these objects. The choice of this set of primitive visualization objects is based on our experience of geometric algorithms, and the set can always be extended and modified.

TABLE I
PRIMITIVE VISUALIZATION OBJECTS CURRENTLY PROVIDED BY THE GEOBUILDER SYSTEM.

Type	Objects	Header and Source Files
2D simple objects	Circle	GeoCircle.h, GeoCircle.c
	Point	GeoPoint.h, GeoPoint.c
	Segment	GeoLineSeg.h, GeoLineSeg.c
	Node	GeoNode.h, GeoNode.c
	Edge	GeoEdge.h, GeoEdge.c
2D composite objects	Triangle	GeoTriangle.h, GeoTriangle.c
	Rectangle	GeoRectangle.h, GeoRectangle.c
	Polygon	GeoPolygon.h, GeoPolygon.c
	Undirected graph	GeoUGraph.h, GeoUGraph.c
	Directed graph	GeoGraph.h, GeoGraph.c
	List	GeoList.h, GeoList.c
3D simple objects	Point	Geo3DPoint.h, Geo3DPoint.c
	Segment	Geo3DLineSeg.h, Geo3DlineSeg.c
3D composite objects	Polygon	Geo3DPolygon.h, Geo3Dpolygon.c
	Polyhedron	Geo3DPolyhedron.h, Geo3DPolyhedron.c
	List	Geo3DList.h, Geo3DList.c

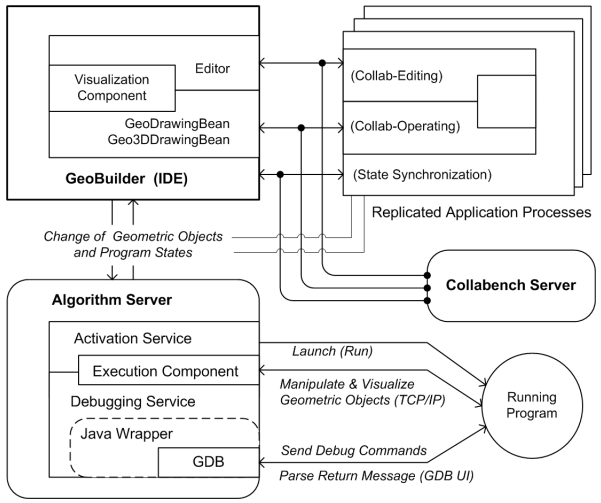


Fig. 2. System Architecture of the GeoBuilder. The backend of the GeoBuilder system connects to an Algorithm Server to request compiling and debugging services. The states of replicated application processes can be synchronized by the Collabench Server.

A. Architecture of GeoBuilder

Based on Java's portability, GeoBuilder is platform independent and can be launched in any web environment. Fig. 2 shows the system architecture of GeoBuilder. The backend of the GeoBuilder system connects to an *Algorithm Server* using TCP/IP sockets. During program development, source programs are submitted to this server. They are then compiled by GNU's *gcc* compiler and launched by the server's *activation service*. The *visualization* and *execution* components provide coupled functions on the two sides for users to interact with the running program; consequently, object-oriented visualization classes allow users to manipulate geometric objects directly via a point-and-click mechanism. Users can further input, save, and restore these geometric objects. The GeoBuilder system also supports a debugging mode, in which the submitted programs are tracked by GNU's *gdb* debugging tool. The Java Wrapper of the Algorithm Server parses and returns the debugging messages for the GeoBuilder IDE to update the current states and watch the internal variables.

GeoDrawingBean and Geo3DDrawingBean are the drawing beans that render geometric 2D and 3D objects respectively. They implement the ActionListener interface as the main part of the visualization component for listening to the execution and debugging messages. These two drawing beans render geometric objects and store them into the GeoModel and Geo3DModel. As for connection to the Collabench server, we developed subclasses inheriting all the above object functions and enabling access to current objects for collaborative work. In case users check the box to enable automatic camera decision for 3D algorithm visualization, Geo3DDrawingBean must forward 3D objects to CameraAnimation, which handles the operation of automatic camera positioning. After finishing the task, CameraAnimation transmits the output objects to Geo3DDrawingBean through the ActionListener interface again.

B. Communication of Geometric Objects between Clients and the Algorithm Server

We use the Java language to develop our system that deals with C/C++ programs in the current stage. The GeoObject class is referenced in users' C++ programs to implement geometric algorithms, while the VisualObject of Java 2D and the GLObject of JOGL will display the GeoObject in a Java-based platform. This implementation implies that we have to coordinate those objects across the C++ and Java languages. Herein, we will introduce both kinds of C++ and Java objects from the viewpoint of their mutual communication. Then we explain how to implement this communication.

1) GeoLEDA library and GeoObject Abstract Class: LEDA [19], [47] is a mathematical library providing efficient data types for combinatorial and geometric computing. Our system supports LEDA 4.2 free version for users' convenience to develop programs. Table I lists the header files and their source files contained in the geoLEDA/ subdirectory [50]. Users must include the needed header files for the data types they would like to use. Corresponding source files will be automatically linked at compile time for user programs to access the objects provided by LEDA library and to communicate with the objects displayed in the Java-based platform. Notably, all the objects contained in these files inherit the above mentioned GeoObject class.

User programs edited in the local GeoBuilder application will be delivered to a remote algorithm server for debugging and execution. Therefore, programs must also call an API function, *IPCServiceSetup()*, to initialize TCP/IP connection before these GeoObjects can be visualized in the GeoBuilder application. This function and other visualization control functions, such as *GeoPause()* and *GeoScanf()*, are called GeoIPC APIs of the GeoLEDA library. In addition to these APIs, other object I/O functions have been directly implemented in the geometric objects classes. GeoBuilder can generate frame codes for users to manipulate geometric objects in their program. The details of geometric object classes can be found in [50]. We will also explain the steps of geometric algorithm developments in Subsection D.

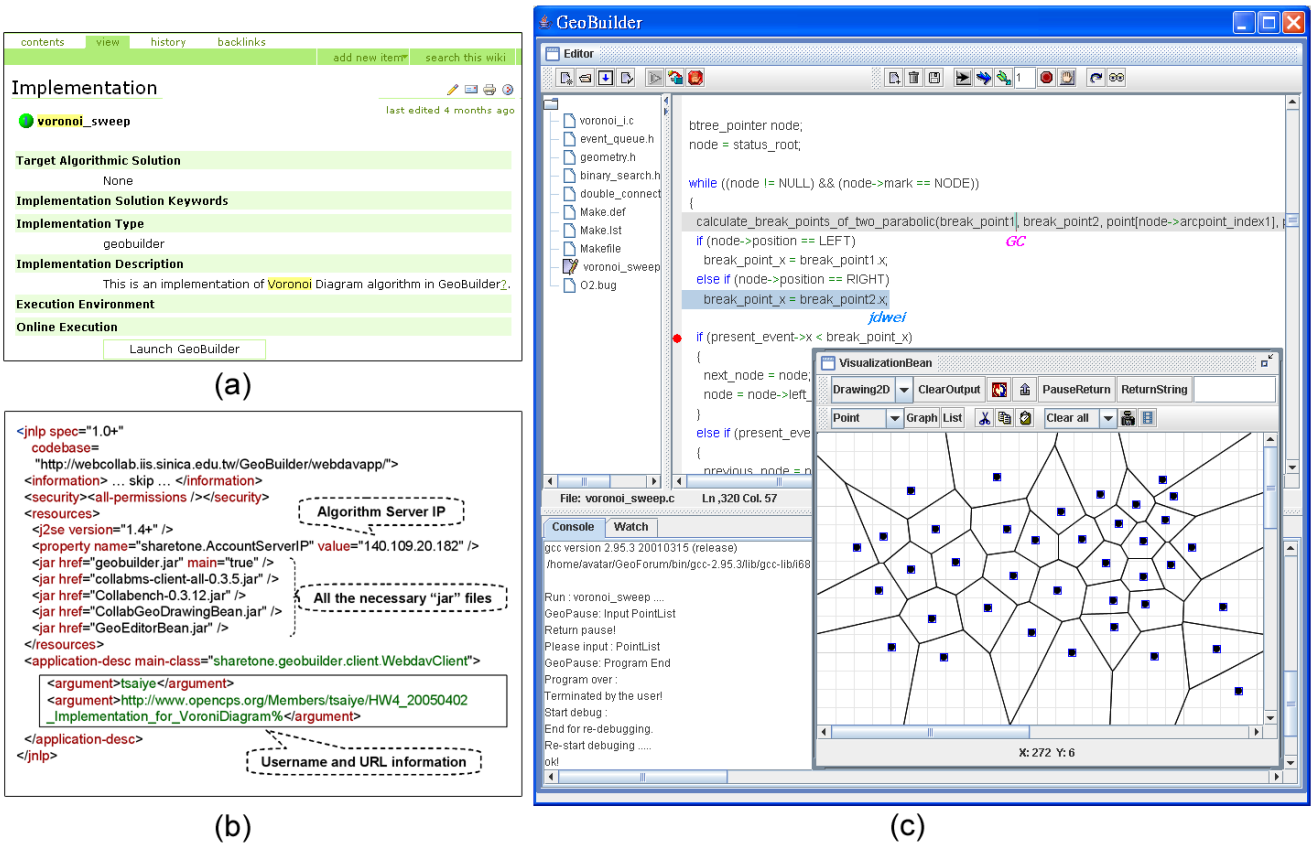


Fig. 3. GeoBuilder is embedded into the OpenCPS portal. (a) GeoBuilder as a content object in the implementation space; (b) Associated JNLP (Java Network Launch Protocol) file; (c) Running a 2D Voronoi diagram program.

2) VisualObject of Java 2D and GLObject of JOGL: The Java classes displaying geometric objects on the screen include the industrial standard Java2D VisualObject class and the JOGL GLObject class for 2D and 3D algorithm visualization respectively. Java 2D was released by Sun Microsystems with the Java 2 SDK, version 1.2, and was broadly used to display 2D objects in graphic Java application. The GeoDrawingBean of GeoBuilder uses the Java 2D VisualObject class corresponding to the C++ GeoObject class. Using the GeoBuilder system, program developers need not know the details about the Java graphic interface. GeoBuilder handles the drawing process synchronously with the change of the C++ GeoObject.

We use the JOGL library for 3D geometric algorithm visualization. JOGL provides hardware-supported 3D graphics to applications written in Java. The technologies of JOGL were officially released as open-source utilities by the Game Technology Group at Sun Microsystems in 2003. JOGL supports full access to the APIs in the OpenGL 2.0 specification as well as nearly all vendor extensions, and integrates with the AWT and Swing widget sets [49]. The Geo3DDrawingBean of GeoBuilder makes use of JOGL to render 3D graphics in a Java-based platform. Geo3DDrawingBean contains the following objects included in JOGL: GL3DPoint, GL3DLineSegment, GL3DPolygon, GL3DPolyhedron, and GL3DList. They inherit the class of GLObject.

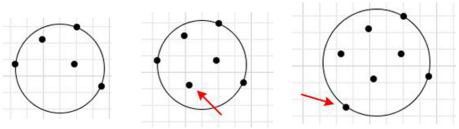
3) Communication between C++ and Java Classes: When users' C++ programs generate, delete, or modify geometric objects inheriting the class of GeoObject, the changes of object states are transformed into string format and then transmitted to the GeoBuilder application through TCP/IP sockets. Therein, the string format is restored to be an instance of the corresponding Java class and forwarded through the early mentioned ActionListener interface to GeoDrawingBean or Geo3DDrawingBean according to the dimension of geometric objects. The corresponding Java objects of VisualObject or GLObject are thereby created, destroyed or updated in response to the state changes. In the opposite direction, the modification of VisualObject and GLObject objects can also be conveyed to users' C++ programs through the string flow on network. Communication like this maintains the objects consistent between C++ GeoObject class and the corresponding Java classes.

C. GeoBuilder as a Plug-in of OpenCPS Knowledge Portal

The OpenCPS knowledge portal was built upon the Plone/Zope open source utilities [52], which support the Python language to develop plug-in packages. Based on its easy plug-in property, we developed a content type running Python scripts in response to web users' implementation and demonstration requests. Figs. 3(a)(b)(c) sequentially show the content object in the implementation space, the JNLP (Java Network Launch Protocol) file, and the demonstration of running a 2D Voroni diagram program.

```
//TestGeometry.c
//Do not modify the auto-generated codes

//beginning of auto-generated codes
#include "TestGeometry.h"
//ending of auto-generated codes

//beginning of auto-generated codes
AlgorithmIO *algorithmIO;
//ending of auto-generated codes

//beginning of auto-generated codes
void TestGeometry_Main(){
    GeoCircle *inCircle = algorithmIO->getInCircle();
    GeoCircle *outCircle = new GeoCircle();
//ending of auto-generated codes

//-------------------------------------------------
//Main routine begins here
//-------------------------------------------------

IPCServiceSetup(char* hostname, int port_id);
//Here is the entry point of your code.

//-------------------------------------------------
//End of main rountine
//-------------------------------------------------

//beginning of auto-generated codes
    algorithmIO->setOutCircle(outCircle);
//ending of auto-generated codes
}
```

(a)



(b) (c) (d)

Fig. 4. Frame code to develop a geometric algorithm with circle objects involved. (a) auto-generated codes; (b) (c) (d) Finding the smallest enclosing circle.

The JNLP file should be prepared in the plug-in package accompanying all Java programs of the GeoBuilder system. This file includes all the information such as the algorithm server's IP address, the launcher's username, the URL of the target program, and all the necessary JAR files of the GeoBuilder system. The JNLP file is for Sun's Java Web Start to launch GeoBuilder in web environment. For the details of Zope configuration and JNLP format, users may refer to the Zope user manual [55] and the guide to Java Web Start [48].

D. Geometric Algorithm Development

Since we have embedded GeoBuilder as a plug-in content type of the portal, users can click any *Algorithm Implementation* content object to start this system. The GeoBuilder system loads or creates project files on the OpenCPS website through the WebDAV protocol. To begin geometric algorithm development, as earlier mentioned in Subsection B, users have to include the appropriate header files for certain data types they would like to use. These files are within the GeoLEDA library and are listed in Table I.

For users' convenience, GeoBuilder provides a frame code generator that asks for the required data types and then creates program files with frame codes in default. Users can thus easily use the GeoLEDA library, as long as they define the input and output in creating a new algorithm. Fig. 4(a) is an example of auto-generated codes to deal with circle objects in the algorithm, *TestGeometry*. The input thereof is a circle and the output is also a circle. The input variable name is $inCircle$ and the output variable name is $outCircle$. This program can be implemented to find the smallest enclosing circle, as shown in Figs. 4(b)(c)(d).

In the above example, three program files are generated in the project: *TestGeometry.c*, *TestGeometry_GeoMain.c*, and *TestGeometry.h*. The first two files contain the main routine and the user procedures respectively. The last file, *TestGeometry.h*, is the overall header file, in which other primitive header files of geometric objects in the GeoLEDA library are also encapsulated. Notice that the sample program calls an API function, *void IPCServiceSetup(char* hostname, int port_id);* in front of the entry point of the main routine. This function initializes a TCP/IP connection to the algorithm server. After the connection is established, the program can thus receive user commands, read input instances, and write the outputs to the canvas panel of the GeoBuilder drawing beans. The arguments of this function indicate the hostname (or IP address) and the port number of the Algorithm Server. They are pre-assigned in the JNLP file, as shown in Fig. 3(b).

IV. Concurrent Collaboration

Coined as a shorthand notion of collaborative workbench, *Collabench* provides a service for collaborative event broadcasting and concurrent action. This service package [54] uses an event-driven framework to maintain a consistent state among replicated applications, facilitating the task of developing concurrently collaborative applications and is used to equip the GeoBuilder system with collaboration capability.

A. Architecture of the Collabench Service Package

When the user creates objects, moves objects or modifies object properties in an application, the application's states will change as a result. The Collabench service allows user applications to broadcast corresponding events to other collaborative applications in the same session, so all the collaborators can see the same changes.

The core idea of Collabench is an event translating mechanism that maps "local events" (LocalEvents) to "collaborative events" (CollabEvents) and vice versa. In the above mentioned situation, relevant local events are fired reflecting the state change made by users' operation on a local object. Using the Collabench service package, application developers wrap local objects (LocalObjects) in collaborative objects (CollabObjects) and then register corresponding LocalEvents that will be translated into CollabEvents.

Fig. 5 shows how the Collabench service works: when the user application calls the $createCollabObject()$ function[3] for creating a CollabObject, a CommonEventListener class for

[3]For wrapping local objects and creating collaborative objects, this function is declared as $CollabenchToolkit.CollabObjectcreateCollabObject(ObjectlocalObjectInstance, Class[\,]localEventClasses)$

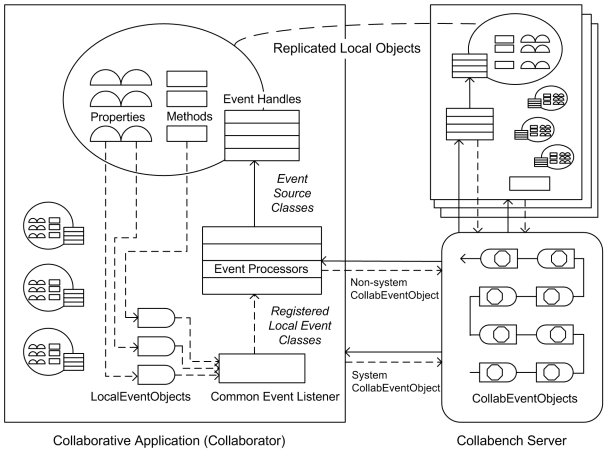


Fig. 5. The infrastructure of the Collabench service. Collabench is an event-driven framework generating and broadcasting collaborative events for concurrent co-operation. This mechanism extracts CollabEventObjects from LocalEventObjects and sends them to the Collabench server (the dashed lines). After receiving the turning-around event objects (solid lines), the EventProcessor then restores the state change from collaborative events to target local objects.

LocalEvents assigned within the CollabObject is automatically created and added into the listener interfaces within the wrapped LocalObject. This proxy class implements all the listener interfaces for the given localEventClasses. The CommonEventListener is instantiated along with a Java invocation handler LocalEventListenerInvocationHandler, which receives LocalEventObjects and extracts CollabEventObjects with the help of EventProcessors.

The client-side *EventProcessor* is programmed by the application developer and bound to a target localEventClass. As Fig. 5 shows, the EventProcessor generates CollabEventObjects to snapshoot the property change or method invocation information originated from the source local objects. These CollabEventObjects are sent to the Collabench server and then dispatched to replicated collaborative applications following the first-come-first-served rule. The respective EventProcessor must also restore the state change from CollabEventObjects to target LocalObjects; therefore, collaboration behaviour is instantiated and maintained.

We further explain the "system CollabEventObject" that appears in Fig.5. The event objects in this class are used to communicate and control the Collabench server, but not to broadcast for concurrent operation. For example, the first launched client in a collaborative session must use *CollabRequestWorkbenchEvent* to request an event queue for the collaboration session. This client application becomes the session owner and must share the current application states and the geometric scene description with the late coming client processes. The latecomers use another system CollabEventObject, *CollabRequestFloorEvent*, to stop broadcasting temporarily and wait to catch up with the current progress. In case a time limit is exceeded, the session owner is responsible to send a CollabEventObject *CollabRefreshWorkbenchEvent* to kick out the latecomers and reactivate concurrent session. Once the owner leaves the algorithm development project, the Collabench server will select another client process to take over the ownership. If all the clients quit this project, the Collabench server will close the project and clear the event queue.

B. Connection between GeoBuilder and Collabench

The connection between GeoBuilder and Collabench is depicted in Figure 2. The text editor, drawing beans, and other objects, such as buttons and checkboxes, in the GeoBuilder IDE are wrapped by respective CollabObjects. When the collaboration mode is enabled, these objects of a GeoBuilder application will be kept consistent with those of all the other replicated GeoBuilder application processes. CollabGeo3DDrawingBean is an example that delegates Geo3DDrawingBean at the front end and takes charge of Collabench workflow to enable collaborative geometric instance editing. Notably, the dynamically created geometric objects at run time during the program execution will inherit VisualObject and GLObject classes in the Java drawing beans. We can therefore add/delete geometric objects or change their properties in one client, and the relevant events will be copied to and reproduced in other GeoBuilder clients in the same collaboration session. The contents of the text editor, the commands to the development environment, the outputs generated step by step by the algorithm server, and the actions of camera, regardless of manual or automatic occurrence[4], are all synchronized among the replicated GeoBuilder clients. Built upon this system architecture, GeoBuilder is inserted in a collaborative integrated programming environment with visualization support.

Fig. 6 shows the state-vector diagram that results when we apply Collabench service to the GeoBuilder system. Figs. 6(a)(b)(c)(d) sequentially demonstrate the following scenarios: the first GeoBuilder application requests an event queue, the second application joins as a latecomer, CollabEvent A is broadcast, and the third application joins the same session. Figs. 6(e) and (f) show two significant design issues. In Fig. 6(e), CollabEvent B and C are broadcast following the first-come-first-served policy. When the Collabench Server broadcasts the collaborative events, the semantics of these events are ignored. If the events conflict in the order in which they come, the GeoBuilder application is responsible for making a reasonable solution. For example, the second and the third applications are moving and deleting the same object in CollabEvents C and B respectively. If event B (deleting) is first broadcast and conducted by all collaborative applications, then event C (moving) must be ignored after it is transmitted to the applications later.

Fig. 6(f) explains what the GeoBuilder application has to do when making compiling/debugging control commands. All the GeoBuilder clients launched in a collaboration session do connect to the algorithm server. To prevent the clients from reduplicating control commands to the algorithm server, all the online IDE operations for compiling, debugging, etc.

[4]The camera positioning mechanism is built inside the Java drawing beans and runs in each individual client process. Different from manual movement, automatic camera positioning only generates one CollabEvent as the enabling signal. This design option reduces the queue length of CollabEventObjects in the Collabench server.

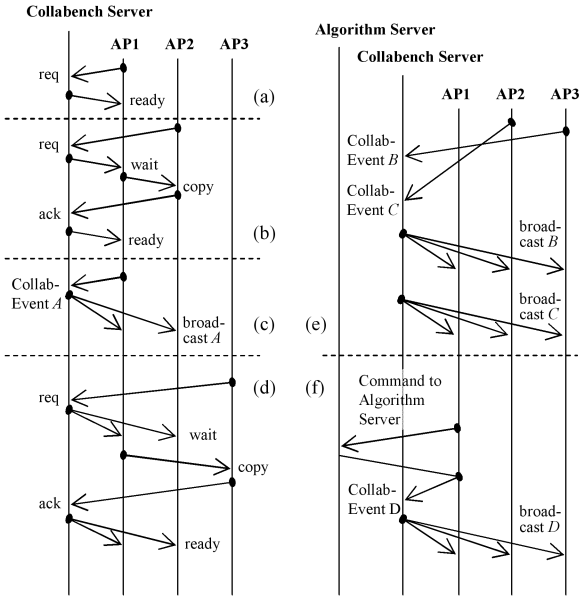


Fig. 6. State-vector diagram illustrating Collabench service. (a) Collabench Server creates an event queue at the first application's request; (b) The second application requests to join this collaboration session; (c) Collabench Server broadcasts CollabEvent A; (d) The third application joins the same session; (e) CollabEvent B and C are broadcast following the first-come-first-served policy; (f) User applications direct their compiling/debugging requests solely to the algorithm server – after the commands are conducted, the state updating information are then broadcast through collaborative events, such as CollabEvent D.

are forked into controller interface events and service request events. The former, showing the button press effect and waiting for a response (if necessary), is wrapped as CollabEvents and broadcast out; on the other hand, the latter must consist of unbroadcast local events that are directed solely to the algorithm server. As Fig. 6(f) shows, the service request events update the application states after making requests, receiving their response, and accordingly triggering output actions. The state updating will subsequently be broadcast through collaborative events, such as CollabEvent D, to other GeoBuilder applications, thus reactivating the collaboration synchronously.

C. Benefits from Concurrent Collaboration

GeoBuilder is a collaborative algorithm development tool that supports geometric algorithm visualization in distributed environments. We have embedded this system into our OpenCPS knowledge portal and used it as an implementation platform for teaching a graduate course, geometric computing and algorithm visualization. Benefits from the collaboration function are clearly observed in the classroom. The feature of concurrent collaboration helps this system reach the highest level of Naps et al.'s criteria of visualization, i.e., presenting, as earlier stated in Section I. The following examples express how the concurrent collaboration could be used.

In office hours, students can invite Teaching Assistants (TAs) to join their collaboration session in order to discuss the problems they have encountered. The conversation part of this discussion may progress over the built-in chat room service, other third-party messaging product, or the traditional telephone line. Importantly, the TA can operate the GeoBuilder system to help students in testing their geometric input instances, verifying their source codes, and even debugging their programs. The concurrent collaboration mechanism makes the TA's extracurricular instructions more efficient than those of other remote desktop products because: (a) GeoBuilder allows multiple users to join a session; (b) the TA can assign several tasks to the group of students and supervise or co-edit with them at the same time; (c) the strongly coupled teamwork mode naturally encourages students to understand their partners' habits of working and thus fosters an atmosphere of tacit agreement that is particularly helpful to collaborative learning and mutual advancement [39]. After a period of practice, some student groups have thrived in this collaborative development environment.

Interestingly, the above process turns around when we need the students to demonstrate their exercise to the TAs. In this case, the TA creates collaboration sessions and invites students to join. As a regular examination, the TA may modify the input instances, insert some program segments for testing, and even ask students to trim the algorithm or add new functions. By using concurrent collaboration, the online demonstration is effective and follows the principles of openness, justice, and fairness.

V. Three Dimensional Drawing

Here we introduce the 3D I/O Mechanism of Geo3DDrawingBean, before we can explain the third feature of the GeoBuilder system, i.e., automatic camera positioning for tracking 3D geometric objects. To place 3D objects using traditional 2D devices is a challenging issue. In this work, we implemented the concept of a *3D cursor* to draw 3D geometric objects.

A. Three-Dimensional Object Input Mechanism

Fig. 7(a) shows the 3D canvas panel in the center, where the long arrows colored in red, green, and blue stand for the x-axis, y-axis, and z-axis respectively. The 3D cursor is crossed by another three arrows that indicate the directions of x-y-z axes. Two labels at the bottom of the 3D canvas panel assist the user in operating the 3D cursor and objects. The camera label displays coordinates of the camera, whereas the object label displays coordinates of the focused 3D cursor and objects. All the coordinates are in double precision in data processing. The yellow coordinate lines are for the users' reference to augment the sense of the 3D cursor position. When creating geometric instances, the user can put the 3D cursor on a sequence of target positions.

The 3D cursor is inherently a point in 3D. Using this special point, Geo3DDrawingBean provides a feasible interface for the user to deal with geometric object classes. The user can operate the mouse and mouse wheel to move the 3D cursor and thus mark a point, draw a line, generate a polygon, or generate a list of these objects. After a 3D object is created, the user can select it again with the 3D cursor. The selected object gets highlighted and the yellow coordinate lines will

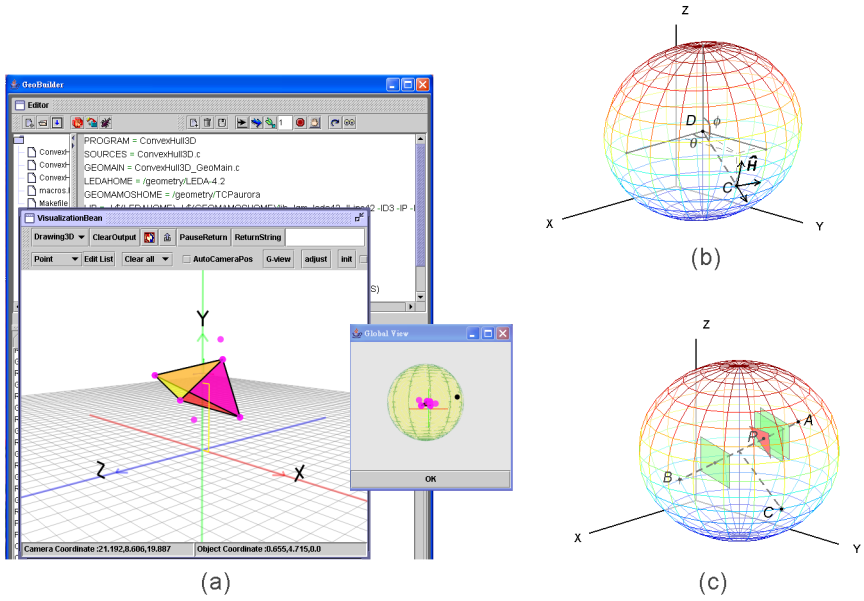


Fig. 7. Observation Sphere and Camera Position Decision. (a) Snapshots of the GeoBuilder IDE, the canvas panel of Geo3DDrawingBean, and the window of global view; (b) Observation sphere and the parameters of the camera position; (c) The camera will move from original position C to a new position to watch the object, centered at P, in the next step. The candidate points A and B are the projections of P onto the observation sphere.

mark out the center of its location. Moving the 3D cursor can also translate the focused 3D objects, together with the associated line segments or polygons.

B. Three-Dimensional Object Output Observation Mechanism

Camera positioning is important in a 3D environment because the geometric objects to be visualized may overlap in front of the camera view. Herein, we explain our design of what we call an *observation sphere* and then present the manual and automatic methods for camera positioning. After creating the input instances, the user must highlight an object or just put the 3D cursor on a final position. The selected position becomes the center of the observation sphere when the user executes the program. Fig. 7(a) illustrates a global view window on the top layer, which shows the observation sphere and pops up along with the main window of Geo3DDrawingBean. When the camera moves to the back of the observation sphere, the sphere becomes transparent to make the camera position visible.

GeoBuilder lays the camera on the observation sphere aiming at the center of the sphere. Therefore, the camera's position is determined by its azimuthal angle, polar angle, and distance to the center of the observation sphere. All three of these arguments can be modified during algorithm visualization, but we need not write a program segment to figure out the relative positions of observed objects for display since the JOGL supports the OpenGL Utility Library (GLU). The $gluLookAt()$ method of the $javax.media.opengl.glu$ class helps us set the global information of the camera that updates the current viewing transformation and drives the Geo3DDrawingBean to redraw the geometric objects immediately.

The parameters of the $gluLookAt()$ method include the position of the camera, the position of the reference point (i.e., the center of the viewport), and the upward vector of the camera. In Fig. 7(b), we let points C and D be the camera position and the center of the observation sphere, where θ and ϕ are the azimuthal and polar angles respectively. Points C and D are the first two sets of parameters of this method. As for the rest of the parameters, we calculate the unit vector, $\hat{H}$, which indicates the upward direction of the camera as follows:

$$\hat{H} = [-cos\theta cos\phi \ \ -sin\theta cos\phi \ sin\phi]^T$$

In other words, we keep the viewpoint of the camera aiming at the center of the observation sphere. In the manual mode, the user can use the arrow keys to move the camera position along the surface of the sphere. While in automatic positioning mode, the drawing bean itself smoothly adjusts azimuthal and polar angles to change the camera position. In both modes, the default radius of the observation sphere is sufficiently large to cover all the input objects, and the user can enter the "+" and "-" keys at any time to alter the radius for zooming in and out respectively.

C. Automatic Camera Positioning

Considering the situation shown in Fig. 7(c), GeoBuilder has calculated the center of the next object of interest as point P while the camera is currently located at position C. In automatic positioning mode, GeoBuilder will move the camera from position C to one of the candidate positions A and B, which are the projections of P onto the observation sphere. Our decision rules to select the new position between positions A and B are ordered as follows: (1) If the vector (A, P) intersects fewer existing objects than (B, P) does, then choose A; otherwise choose B; (2) If we cannot make a decision by the first rule, the nearest point to the current position C will be chosen. The procedure in each iteration for showing a new object includes the following six steps:

1) Calculate the center of the next object of interest;

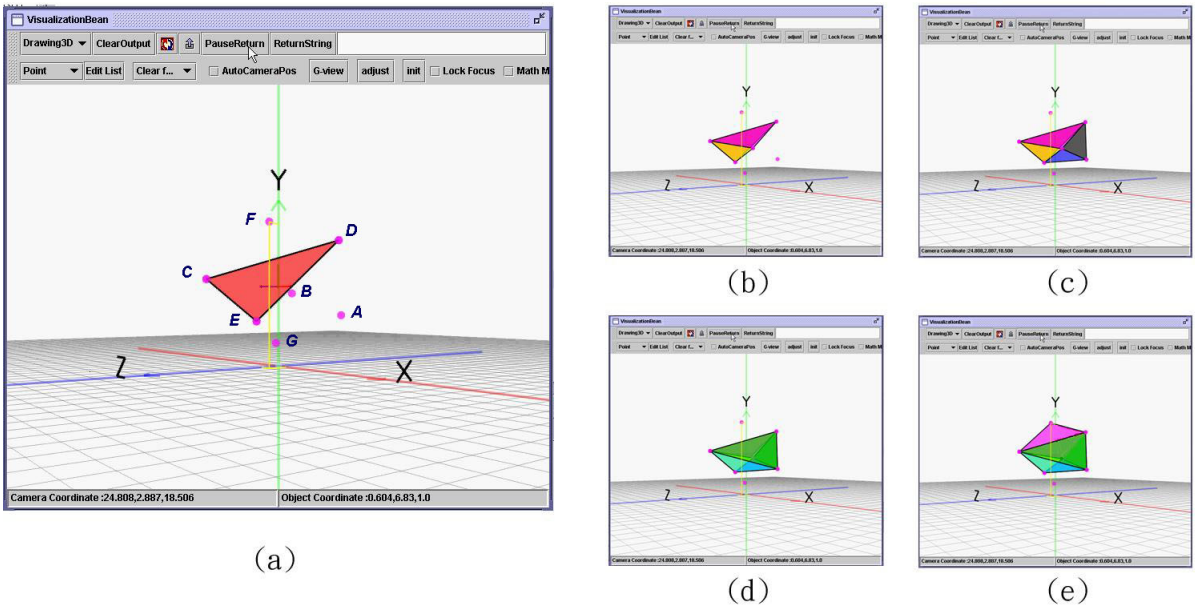


Fig. 8. Snapshots captured while constructing the convex hull. This procedure is visualized with a camera unmoved in the manual mode; seven input points from A to G sequentially locate farther away from the viewport. The five subfigures illustrate in order the triangles generated: (a) ΔCDE, (b) ΔBCD and ΔBCE, (c) ΔABD and ΔABE, (d) ΔACD, and (e) ΔCDF. Notably, the 3D objects may overlap one another in front of the camera. Without moving the camera dynamically to proper positions, we cannot observe all changes of algorithm states.

2) Decide the next camera position;
3) Move the camera smoothly along the shortest path on the spherical surface. In doing so, the 3D drawing bean keeps getting redrawn at an acceptable speed;
4) Display and highlight the object of interest;
5) Allow the user to change the camera position and the radius of observation sphere manually. The updated camera position and radius of the sphere are saved, such as it is done by automatic decision;
6) Turn off the highlights and go into the next iteration of geometric algorithm visualization.

We have presented a video of the automatic camera positioning in the 22nd Annual ACM Symposium on Computational Geometry (SoCG 06) [45]. This video shows how we launch the GeoBuilder on the OpenCPS website in the first step. Two 3D algorithms about constructing convex hulls [4] and detecting line segment intersections [33] are then loaded to conduct the algorithm visualization.

D. Demonstrations of Automatic Camera Positioning

Figs. 8 and 9 show the snapshots when we construct a convex hull in the manual and automatic positioning modes respectively. In Fig. 8, the snapshots are captured with a camera unmoved in the manual mode. Five subfigures illustrate in order the generated hull triangles. Observably, these objects are likely to overlap one another in front of the camera when we do not move the camera dynamically to proper positions. Fig. 9 shows the snapshots captured while constructing the convex hull with automatic camera positioning. In this mode, the camera is automatically moved to visualize the algorithm step by step. Comparison between these two figures exposes that automatic camera positioning provides a clearer effect in visualization procedure.

Geo3DDrawing bean is also suited for line segment associated algorithms. Fig. 10 demonstrates the animation in detecting line segment intersections in a 3D environment. This algorithm repeats focusing on one line segment and sequentially examining potential intersections with other line segments. During this procedure, the next objects of interest are alternately the tested line segment and the nearest point thereon towards the focused line segment. The intersection point can thereby be clearly observed, if it actually exists.

We are finding the optimum sequence of camera positions in this work; however, moving the camera automatically is not always powerful enough to present the algorithm states. In certain situations, the changes to the scene occur in many different places. If there is only one camera, the camera will keep moving and the user will lose track of where things are. In future work, we will provide multiple views of visualization. Each view window will have its own camera, and the user can dynamically determine the number of views for algorithm visualization. We continue the idea of automatic camera positioning in this work, but only the camera that is closest to the next object of interest will move to the new position. This sort of design will reduce disorienting motion.

VI. Conclusion

In this paper, we have presented a geometric algorithm visualization system, GeoBuilder, which features Java's portability, concurrent collaboration, and the dynamic decision of camera position for 3D geometric algorithm visualization. From the viewpoint of Naps et al's criteria, the GeoBuilder system can reach the highest level of learner engagement, i.e. the

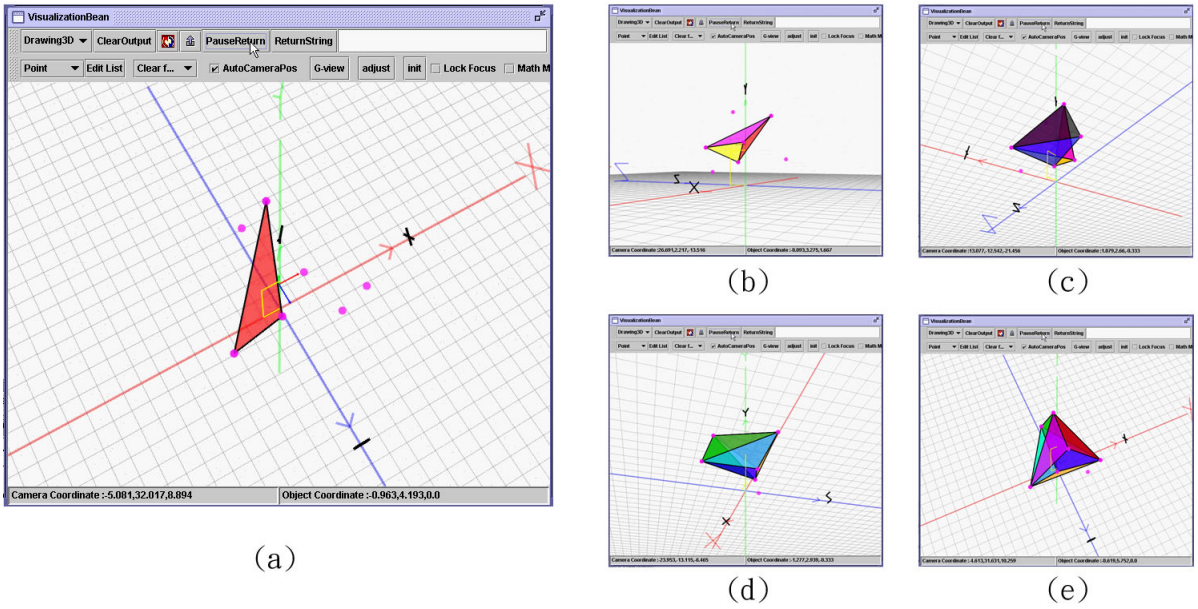


Fig. 9. Snapshots captured while constructing the convex hull with automatic camera positioning. The camera in this mode is automatically moved to visualize the algorithm step by step. In comparison with Fig. 8, the corresponding subfigures from (a) to (e) show the runtime states more clearly in generating hull triangles.

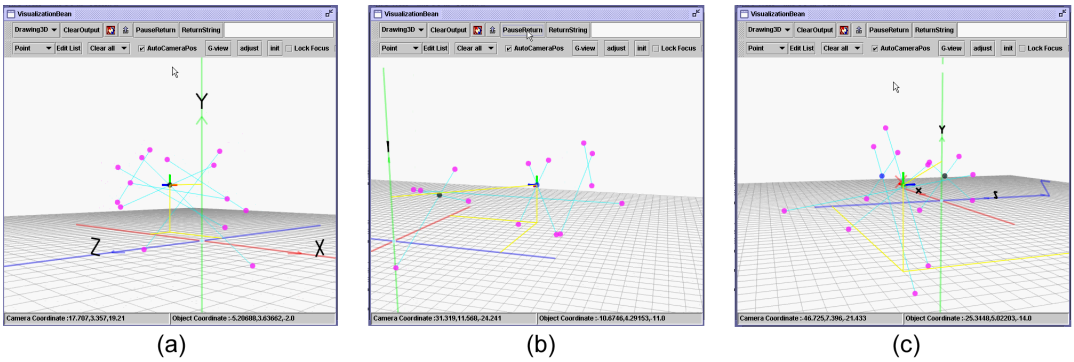


Fig. 10. GeoBuilder can also visualize line segment associated algorithms. Subfigures demonstrate the animation in detecting line segment intersections in a 3D environment.

presenting level. This achievement mainly benefits from the system portability, allowing the system itself to be plugged into a knowledge portal. We have explained how to embed GeoBuilder into our previously developed OpenCPS portal. The users can exchange comments after they construct their own programs. Furthermore, the collaboration capability also provides a useful channel for the users to demonstrate their new ideas and findings in an online manner.

A good algorithm visualization system must address various features meeting the users' need, and its development requires implementation of a variety of modules integrated together. We have proposed a camera positioning module, in addition to addressing the system portability issue and collaboration capability, because animation effect is always the kernel issue of an algorithm visualization system, and other considerations such as platform independence and capabilities to allow teamwork are also important. Although our camera positioning rule is simple, it applies well to automatically track 3D geometric objects as shown by two examples of 3D algorithms, i.e., constructing convex hulls [4] and detecting line segment intersections [33].

A. Engagement in a Collaborative Learning Environment

Felder in 1993 indicated that students have their own learning styles. These styles can be categorized into eight modes as shown in Table II. There are no correct but only preferred learning styles; furthermore, students may prefer one side of a dimension in some subjects and the opposite side in other subjects [21]. Traditional coursework puts particular stress on the dimensions of learning on the left side, i.e., students are taught to think about the concepts on the lectures step by step. Felder's learning style of information advises the teachers to reach all the different learning dimensions in a course. To accommodate all the learning styles, teachers can assign hands-on exercises, review the course with a global

TABLE II
FELDER'S LEARNING STYLE

Dimensions	Definitions		Dimensions
Reflective	Think about it	Do it	Active
Intuitive	Learn Concepts	Learn Facts	Sensing
Verbal	Require Lecture	Require Picture	Visual
Sequential	Step by Step	Big Picture	Global

picture, and even use some visualization tools. The combination of GeoBuilder and the OpenCPS portal supports a large set of these facilities. Students can develop their programs for submission and visualize their execution. Presentations following the knowledge management policy of the OpenCPS build a global view of the problems and solutions. The overall system is an effective supplementary tool that is promising for academic research and in educational settings.

B. Discussion of the System Effectiveness

The effectiveness of an algorithm animation system has been studied and analyzed in the field of computer science education. Stasko et al. conducted an empirical study of a priority queue algorithm animation and found that algorithm animation was not necessarily a better way to teach an algorithm [41]. Nevertheless, Lahtinen et al. indicated that all the user groups, teaching materials and classroom instructions can affect the effectiveness of an algorithm visualization system [22], [23]. Developed for geometric algorithm visualization, the GeoBuilder system is especially useful in debugging and education since geometric objects are naturally suitable for users to see the shapes rather than to see the numbers on a printout. The 2-opt algorithm to solve the traveling salesman problem and the quick-hull algorithm to find a convex hull are examples for which we can clearly display the next operations running in the algorithms, i.e., detecting a path intersection and inserting a new edge respectively. The visualization interface can not only help the users to verify their idea but also teach the students to understand the pseudocodes of the algorithms.

The performance issues we have presented in this work involve system portability, concurrent collaboration, and improvement of animation effect. The discussion with regard to these issues has been given in the relevant sections. From the viewpoint of system effectiveness, we conclude: (a) Using the Java's portability, we have demonstrated the integration with our system and a knowledge portal. This integration enhances the application of a visualization system. (b) The concurrent collaboration function is now helpful for TAs to use to guide and test the students. Some students also enjoy this collaborative development environment. We intend to extend this mechanism to a larger scale program development. (c) We have proposed a method for positioning the camera properly to visualize the geometric objects, but moving the camera automatically may not always work in certain cases that the changes to the scene occur on many different places. To solve this problem, our system will support multiple views of visualization in the future.

C. Future Work

It is difficult to have an algorithm visualization system that meets the demand of the user in every aspect. Although GeoBuilder is equipped with a powerful drawing engine for use in 3D environment, there is room for improvement and a lot more work to enhance its capabilities, including but not limited to the following:

1) Consider heuristic methods developed in artificial intelligence and computer vision to obtain more robust decision rules for automatic camera positioning;
2) Combine other multimedia techniques and auxiliary I/O devices to improve the effectiveness of algorithm visualization and facilitate the application in heterogeneous platforms;
3) Follow the open standards XML-enabled 3D file format, the X3D (previously called VRML200x), to enable real-time communication of 3D data across all applications and network applications;
4) Create multiple views of visualization and extend the application of 3D input/output mechanisms to the research of Virtual Reality;
5) Provide more complex 3D geometric object type for use in CAD (Computer-Aided Design) and CAM (Computer-Aided Manufacturing);
6) Develop high-level collaboration scenarios and environments for use in collaborative learning.

ACKNOWLEDGMENT

This work was supported in part by the National Science Council under the Grants NSC96-2221-E-001-016-MY3 and NSC-96-2752-E-002-005-PAE.

REFERENCES

[1] A. Aurum, P. Parkin, and K. Cox. Knowledge management in software engineering education. In *Proc. IEEE International Conference on Advanced Learning Technologies*, pages 370–374. 2004.

[2] W. Bangerth, R. Hartmann, and G. Kanschat. `DEAL.II` *Differential Equations Analysis Library, Technical Reference*. `http://www.dealii.org`.

[3] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II—a general-purpose object-oriented finite element library. *ACM Transations on Mathematical Software*, 33(4):24, 2007.

[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[5] W. H. Bares and J. C. Lester. Intelligent multi-shot visualization interfaces for dynamic 3D worlds. In *Proc. 4th international conference on Intelligent user interfaces*, pages 119–126. 1999.

[6] M. Bäsken and S. Näher. GeoWin– a generic tool for interactive visualization of geometric algorithms. In *Software Visualization*, pages 88–100. Springer-Verlag, 2002.

[7] M. H. Brown and M. A. Najork. Algorithm animation using 3D interactive graphics. In *Proc. 6th Annual ACM Symposium on User Interface Software and Technology*, pages 93–100. 1993.

[8] M. H. Brown and R. Sedgewick. A system for algorithm animation. In *Proc. 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 177–186, 1984.

[9] G. Cattaneo, P. Faruolo, U. F. Petrillo, and G. F. Italiano. JIVE: java interactive software visualization environment. In *Proc. IEEE Symposium on Visual Languages and Human Centric Computing*, pages 41–43, 2004.

[10] P. D'Ambra, M. Danelutto, D. di Serafino, and M. Lapegna. Advanced environments for parallel and distributed applications: a view of current status. *Parallel Comput.*, 28(12):1637–1662, 2002.

[11] S. Diehl. Software visualization. In *Proc. 27th International Conference on Software Engineering*, pages 718–719. 2005.

[12] H. P. Dommel and J. J. Garcia-Luna-Aceves. Floor control for multimedia conferencing and collaboration. *Multimedia Systems*, 5(1):23–38, 1997.

[13] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. *Computer Graphics Forum*, 19(2):101–110, 2000.

[14] J. Fontana. Collaborative software ages slowly. In *Network World Fusion (http://www.networkworld.com/)*, 2003.

[15] A. Hausner and D. P. Dokkin. GAWAIN: Visualizing geometric algorithms with web-based animation. In *Proc. 14th Annual Symposium on Computational Geometry*, pages 411–412. 1998.

[16] T. D. Hendrix, J. H. Cross II, and L. A. Barowski. An extensible framework for providing dynamic data structure visualizations in a lightweight IDE. In *Proc. 35th SIGCSE Technical Symposium on Computer Science Education*, pages 387–391. 2004.

[17] S. Hupfer, L.-T. Cheng, S. Ross, and J. Patterson. Introducing collaboration into an application development environment. In *Proc. 2004 ACM Conference on Computer Supported Cooperative Work*, volume 6, pages 21–24, 2004.

[18] A. Kerren and J. T. Stasko. *Algorithm Animation – Introduction*, chapter 1, pages 1–15. Springer Berlin / Heidelberg, 2002.

[19] L. Kettner and S. Näher. Two computational geometry libraries: LEDA and CGAL. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 64, pages 1435–1463. CRC Press LLC, Boca Raton, FL, second edition, 2004.

[20] M. Knepley, R. Katz, and B. Smith. Developing a geodynamics simulator with PETSc. In A. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 413–438. Springer-Verlag, 2006.

[21] M. Krebs, T. Lauer, T. Ottmann, and S. Trahasch. Student-built algorithm visualizations for assessment: flexible generation, feedback and grading. In *Proc. 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 281–285. 2005.

[22] E. Lahtinen and T. Ahoniemi. Annotations for defining interactive instructions to interpreter based program visualization tools. *Electronic Notes in Theoretical Computer Science*, 178:121–128, 2007.

[23] E. Lahtinen, H.-M. Jarvinen, and S. Melakoski-Vistbacka. Targeting program visualizations. In *Proc. 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 256–260, 2007.

[24] D. Lee, C. F. Shen, and S. M. Sheu. Geosheet: A distributed visualization tool for geometric algorithms. *Internal Journal of Computational Geometry and Applications*, 8(2):119–155, 1998.

[25] D. T. Lee, G. C. Lee, and Y. W. Huang. Knowledge management for computational problem solving. *Journal of Universal Computer Science*, 9(6):563–570, 2003.

[26] L. H. Lin, D. T. Lee, and K. F. Aoki. DAViD: A distributed algorithm visualization and debugging system for geometric computing in 3D. *Journal of Three Dimensional Images*, 15(1):67–73, 2001.

[27] Y. L. Lin, J. D. Wei, G. C. Lee, and D. T. Lee. A visualization tool for the sitemap of a knowledge portal and the concept map of group knowledge. In *Proc. 5rd International Conference on Knowledge Management*, pages 179–186. 2005.

[28] S. Markus, S. Weerawarana, E. N. Houstis, and J. R. Rice. Scientific computing via the web: The net pellpack PSE server. *IEEE Computational Science and Engineering*, 4(3):43–51, 1997.

[29] M. MU. PDE.Mart: A network-based problem-solving environment for PDEs. *ACM Transactions on Mathematical Software*, 31(4):508–31, 2005.

[30] M. A. Najork. Web-based algorithm animation. In *Proc. 38th Conference on Design Automation*, pages 506–511. 2001.

[31] T. Naps. Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, 35(2), 2003.

[32] T. M. Nina Amenta, Stuart Levy and M. Phillips. Geomview: a system for geometric visualization. In *Proc. 11th Annual Symposium on Computational Geometry*, pages 412–413. 1995.

[33] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

[34] P. S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[35] T. Panas, R. Lincke, and W. Lowe. Online-configuration of software visualizations with Vizz3D. In *Proc. 2005 ACM Symposium on Software Visualization*, pages 173–182, 2005.

[36] M. Pendergast. *Groupgraphics: Prototype to product*, pages 209–227. McGraw-Hill International, UK, 1995.

[37] M. Roseman and S. Greenberg. GroupKit: a groupware toolkit for building real-time conferencing applications. In *Proc. 1992 ACM Conference on Computer Supported Cooperative Work*, pages 43–50, 1992.

[38] M. Roseman and S. Greenberg. Building real-time groupware with GroupKit, a groupware toolkit. *ACM Transactions on Computer–Human Interaction*, 3(1):66–106, 1996.

[39] N. Rummel, H. Spada, and S. Hauser. Learning to collaborate in a computer-mediated setting: observing a model beats learning from being scripted. In *Proc. 7th International Conference on Learning Sciences*, pages 634–640, 2006.

[40] M. Shneerson and A. Ta. GASP-II – a geometric algorithm animation system for an electronic classroom. In *Proc. 14th Annual Symposium on Computational Geometry*, pages 405–406. 1998.

[41] J. Stasko, A. Badre, and C. Lewis. Do algorithm animations assist learning?: an empirical study and analysis. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pages 61–66, 1993.

[42] J. T. Stasko and J. F. Wehrli. Three-dimensional computationvisualization. In *Proc. IEEE Symposium on Visual Languages*, pages 100–107. 1993.

[43] C. Sun and D. Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer–Human Interaction*, 9(1):1–41, 2002.

[44] A. Tal and D. Dobkin. Visualization of geometric algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):194–204, 1995.

[45] M.-H. Tsai, J.-D. Wei, J.-H. Huang, and D. T. Lee. A portable geometric algorithm visualization system with dynamic camera positioning for tracking 3D objects. In *Proc. 22nd Annual Symposium on Computational Geometry*, pages 479–480, 2006.

[46] M. Zancanaro, O. Stock, and I. Alfaro. Using cinematic techniques in a multimedia museum guide. In *Proc. Museums and the Web 2003*.

[47] Algorithmic Solutions Software GmbH LEDA. http://www.mpi-sb.mpg.de/LEDA/.

[48] Java Web Start Technology. http://java.sun.com/products/javawebstart/.

[49] JOGL API Project. https://jogl.dev.java.net/.

[50] Manual of GeoLEDA Library. http://webcollab.iis.sinica.edu.tw/ Components/GeoBuilder/ GeoLedaManual/index.html.

[51] OpenCPS Website. http://www.opencps.org/.

[52] Plone Foundation. http://plone.org/.

[53] ShareTone Project. http://webcollab.iis.sinica.edu.tw/SHARETONE, http://www.sharetone.org/.

[54] WebCollab – Collabench's Home. http://webcollab.iis.sinica.edu.tw/ Components/Collabench.

[55] Zope Corporation. http://www.zope.org/Documentation/.