A Portable Geometric Algorithm Visualization System with Dynamic Camera Positioning for Tracking 3D Objects *

Ming-Hung Tsai†, Jyh-Da Wei, Jeng-Hung Huang and D. T. Lee‡
Institute of Information Science, Academia Sinica
128, Section 2, Academy Road, IIS, Nankang, Taipei, Taiwan, 115
{mhtsai, jdwei, jhh, dtlee}@iis.sinica.edu.tw

ABSTRACT

Geometric algorithm visualization techniques are very important to algorithmic research in geometric computing. The relevant applications include geometric algorithm development, testing, demonstration and teaching. Within these topics, there still remain performance issues to improve system portability, animation effect and so on. In this paper, we present a geometric algorithm visualization system that is featured by Java's portability and the "dynamic decision of camera position" for 3D geometric algorithm visualization.

Categories and Subject Descriptors:
I.3.5 [Computational Geometry and Object Modeling] : Geometric algorithms, languages, and systems

General Terms:
Algorithms

Keywords:
Computational Geometry, Algorithm Visualization, Camera Position, LEDA, CGAL, Convex Hull, Line Segment Intersection

1. INTRODUCTION

Geometric algorithm visualization is a unique research topic integrating various engineering skills, such as computer graphics, system programming, database management, computer networks, etc., to facilitate algorithmic researchers in testing their ideas, demonstrating their findings, and teaching in the classroom. In this paper, we present a geometric algorithm visualization tool, "GeoBuilder", which possesses two important features making it more promising than other existing visualization systems. First, the 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, the 3D geometric drawing bean of this system can dynamically decide the camera position for users to effectively track the 3D objects during algorithm visualization. These two features will be further explained in Sections 2 and 3 respectively. We finally express our video sequence in Section 4.

*This work was supported in part by the National Science Council under the Grants NSC94-2213-E-001-004 and NSC-94-2752-E-002-005-PAE.
† ‡ The authors are also with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

SCG'06, June 5–7, 2006, Sedona, Arizona, USA.
ACM 1-59593-340-9/06/0006.

2. SYSTEM ARCHITECTURE

GeoBuilder is rebuilt from our prior system, "GeoSheet" [5]. We create the GeoBuilder system as an Integrated Development Environment (IDE) with full support of LEDA and CGAL [4] libraries 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 such as points, line segments, polygons, and the list of these objects.

The backend of the GeoBuilder system connects to an "Algorithm Server" using TCP/IP sockets. During the algorithm development, source programs are submitted to this server. They are then compiled and tracked by GNU's "gcc" compiler and "gdb" debugging tool respectively. The Java Wrapper of the Algorithm Server parses and returns the debugging message for the Java drawing bean to display the current states of geometric objects. Object-oriented visualization classes allow users to manipulate geometric objects directly via point-and-click mechanism. Users can input, save, and restore these geometric objects. They can also execute one program repeatedly with different input instances.

Based on Java's portability, GeoBuilder is a platform-independent system and can be installed and launched in any web environment. We have implemented this system as a plug-in module embedded in our OpenCPS (Open Computational Problem Solving, *http://www.opencps.org*) knowledge portal [6, 7] and used it as a practice platform for lecturing on "geometric computing and algorithm visualization". GeoBuilder gives students a true insight to see how particular algorithms solve geometric problems and therefore impresses the learning effect.

3. THREE DIMENSIONAL DRAWING

The 3D drawing bean of the GeoBuilder system uses a concept of "3D cursor" to manipulate the geometric objects. Users operate the mouse and mouse wheel to move the 3D cursor and thus to mark a point, draw a line, generate a polygon, or translate/delete a set of objects. After creating the input instances, users must highlight an object or just put the 3D cursor on a final position. The selected position

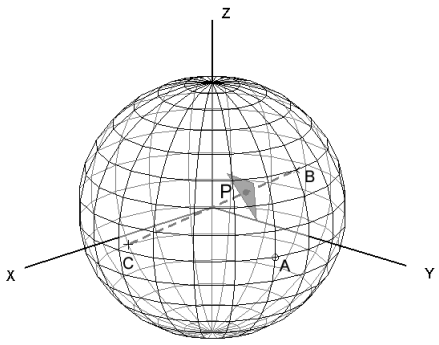

Figure 1: Observation Sphere and Camera Position Decision. The camera will move from the current position A to a new position B or C to watch the object, centered at P, in the next step.

becomes the center of the "observation sphere" when users run the program. Figure 1 shows a situation that we finally move the 3D cursor to the origin. The default radius of the observation sphere is sufficiently large to cover all the input objects. GeoBuilder lays the camera on the observation sphere aiming at the center of the sphere. Users can manually use the arrow keys to move the camera position along the surface of the sphere and enter the "+" and "-" keys to change its radius for zooming in and out.

"Dynamic camera positioning" has received increasing attention in the fields of virtual reality [2, 9] and computer vision [3]. The GeoBuilder system implements this concept for 3D geometric algorithm visualization. Camera position is not a problem in 2D environments. This issue becomes serious in 3D environments, however, because the geometric objects may overlap in front of the camera view. Therefore, we need a decision rule moving the camera to a proper position to focus on the objects in the next step. Consider the situation shown in Fig. 1. GeoBuilder has calculated the center of next object of interest as point P, while the camera is currently located at position A. The camera will be moved from position A to one of the candidate positions B and C, which are the projections of P on the observation sphere. Our decision rules to select the new position between B and C are ordered as follows: (1) If the vector (B, P) intersects fewer existing objects than (C, P) does, then choose B; otherwise choose C; (2) If we cannot make decision by the first rule, the nearest point to the current position A will be chosen.

The procedure of each iteration for showing a new object includes the following six steps:

1. Calculate the center of the next concerned object;
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 redrawn in an acceptable speed;
4. Display and highlight the object of interest;
5. Allow the users to change the camera position and the radius of observation sphere manually. The update of camera position and the radius of the sphere is saved like what is done by automatic decision;
6. Turn off the highlight and go into the next iteration of geometric algorithm visualization.

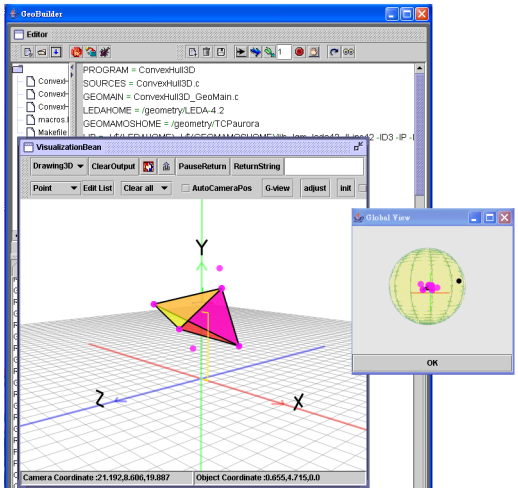

Figure 2: Snapshot of constructing a 3D convex hull.

4. THE VIDEO

The accompanying video shows how we launch the GeoBuilder on the OpenCPS website in the first step. Two 3D algorithms about constructing convex hulls [1] and detecting line segment intersections [8] are then loaded to visualize 3D geometric computing. Besides the main window of the Java drawing bean, a global view involving the observation sphere and camera position is also popped up. We demonstrate both automatic and manual camera positioning in 3D geometric algorithm visualization. Figure 2 shows a snapshot when we construct a convex hull halfway on the video.

5. REFERENCES

[1] 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.

[2] 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 (IUI'99)*, pages 119–126. 1999.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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