

Journal of Mathematics, Statistics and Allied Fields

Volume 1, Issue 1, 2007

Visualization of Algorithms from Computational Geometry using QuickTime® Movies

Jay Martin Anderson: Franklin & Marshall College and Free University of Bolzano-Bozen,
jay.anderson@fandm.edu, <http://www.fandm.edu/x5014.xml>

Abstract

QuickTime®¹ movies provide students of computational geometry with a useful pedagogical tool that helps them to follow, implement and analyze algorithms. QuickTime movies can include explanatory text to support the visualization, breakpoints to stop the visualization for student interaction, and the ability to rewind, fast-forward and move step-by-step through the visualization. QuickTime Virtual Reality movies provide an additional element of interactivity. We introduce here a suite of visualizations spanning many algorithms for many topics in computational geometry, offering thereby to stimulate discussion and debate on the pedagogical merits of this technique.

Introduction

We² have sought to develop a useful teaching accompaniment in computational geometry by constructing animations or visualizations of geometric constructions and data structures as they change during the course of algorithms from computational geometry. The purpose is to illustrate in a way impossible in the print medium, including the classroom blackboard, the operation of these algorithms. In many cases, students in computational geometry and computer graphics have learned to develop the visualizations themselves.

Rather than consider just one problem from computational geometry, we have attempted to develop visualizations for many algorithms for many problems in the subject, using two leading textbooks as guides^{3 4}. We have not constructed visualizations of all algorithms in both textbooks; we have chosen those which are most likely to occur in an undergraduate course or seminar in computational geometry, and those which exhibit some challenging geometric or computational problems.⁵

In what follows we present a brief survey of algorithm visualization as applied in instruction in university-level courses; our workflow for constructing these visualizations; our suite of visualizations for computational geometry; and some thoughts on evaluation and future work.

Algorithm Visualization in Computational Geometry

Algorithm visualization is certainly not new, and debates and discussions on technique, platform and pedagogical effectiveness have occupied practitioners since Brown's landmark thesis⁶ in 1988. Stasko's⁷ book continued the development of ideas in algorithm visualization. "Working Groups" at several consecutive *ITiCSE* conferences attest to the importance of this effort in computer science education.⁸

The ACM review of computational geometry⁹ shows videos in a variety of formats to illustrate the solution to particular problems in computational geometry. Some of these videos include a voiceover to explain the algorithm; many contain text; most are animations of PowerPoint® presentations, and are animations in this sense only.

Hope College has maintained what it calls "the complete collection of algorithm animations."¹⁰ The construction of the convex hull in two and three dimensions, the construction of the Voronoi diagram and a Delaunay triangulation are among the common algorithms visualized in this collection. Many of these visualizations are implemented as Java applets; the visualizations usually have no accompanying text, instructions, or source code.

Some developers have implemented a system for constructing visualizations, such as Stasko's XTango¹¹ or Rößling's Animal¹². Often these systems are confined to a particular platform or operating system and are not easily portable.

Anderson and Naps¹³ have suggested that these seven features of a visualization contribute to achieving student learning: passive observation, explanatory text, the ability to choose input (parameters, initial conditions), breakpoints at important events in the lifetime of the algorithm, appropriate choice of content, the ability to rewind, fast-forward and move step-by-step through the visualization, and the opportunity to design the visualization. It is our assertion that the QuickTime movie provides all of these features except the ability to choose inputs.

In addition, QuickTime has been stable over nearly two decades, is supported on the Mac OS as well as on a variety of Windows operating systems, and is easily "localized" to provide movies in languages other than those of the implementor. The software and software development kits are provided free by Apple for both Macintosh and Windows platforms. Consequently, we have chosen to visualize algorithms from computational geometry using QuickTime movies.

A major disadvantage of QuickTime movies is that the student or viewer has no choice of initial conditions or parameters governing the algorithm. The student must watch the movie that the instructor has provided. The only way to overcome this disadvantage is for the instructor to provide many movies which span the parameter space and allow the student to choose among movies when she cannot choose among parameters.

A Workflow for Creating QuickTime Movies for Algorithm Visualization

In this section, we describe our use of the model-view-controller paradigm for writing computer programs to create movies. We use the example of constructing the Lorenz attractor curve to show how we integrate video and text in a movie and how we create a QuickTime virtual reality movie of a curve in three dimensions. Finally, we describe how we embellish the movies with titles and questions to the viewer at breakpoints, and we acknowledge the additional software used to accomplish these embellishments.

Model-View-Controller paradigm. We have developed and now adhere to a model-view-controller paradigm¹⁴ for writing programs which create QuickTime movies of geometric data structures. In this context, the *model* is the geometrical objects to be visualized; the *view* contains general information about the drawings; and the *controller* deals with QuickTime and the operating system. It is possible to provide a sample package which contains a functional controller requiring very little change from problem to problem, a view which allows simple, intuitive, often cosmetic changes at the desire of the developer; and a model for a very simple problem. The new developer, of course, must implement the model for his own algorithm.

The Model, View and Controller send each other messages. Chief among these messages, in our application, are those which access or change a state variable. The state variable is analogous to the act or scene in a play, and describes a part of the algorithm in which one operation takes place, even if many times; or in which one particular view of the data structure is shown. For example, in one state of the system, the Lorenz curve grows as time moves forward; in another state, a particular Lorenz curve is viewed from many angles. The state variable can be used by the Controller to modify the frame rate of the movie and to stop the movie altogether. The state can be used by the View to set parameters for drawing. The relationship between Model, View, and Controller is shown in Figure 1.

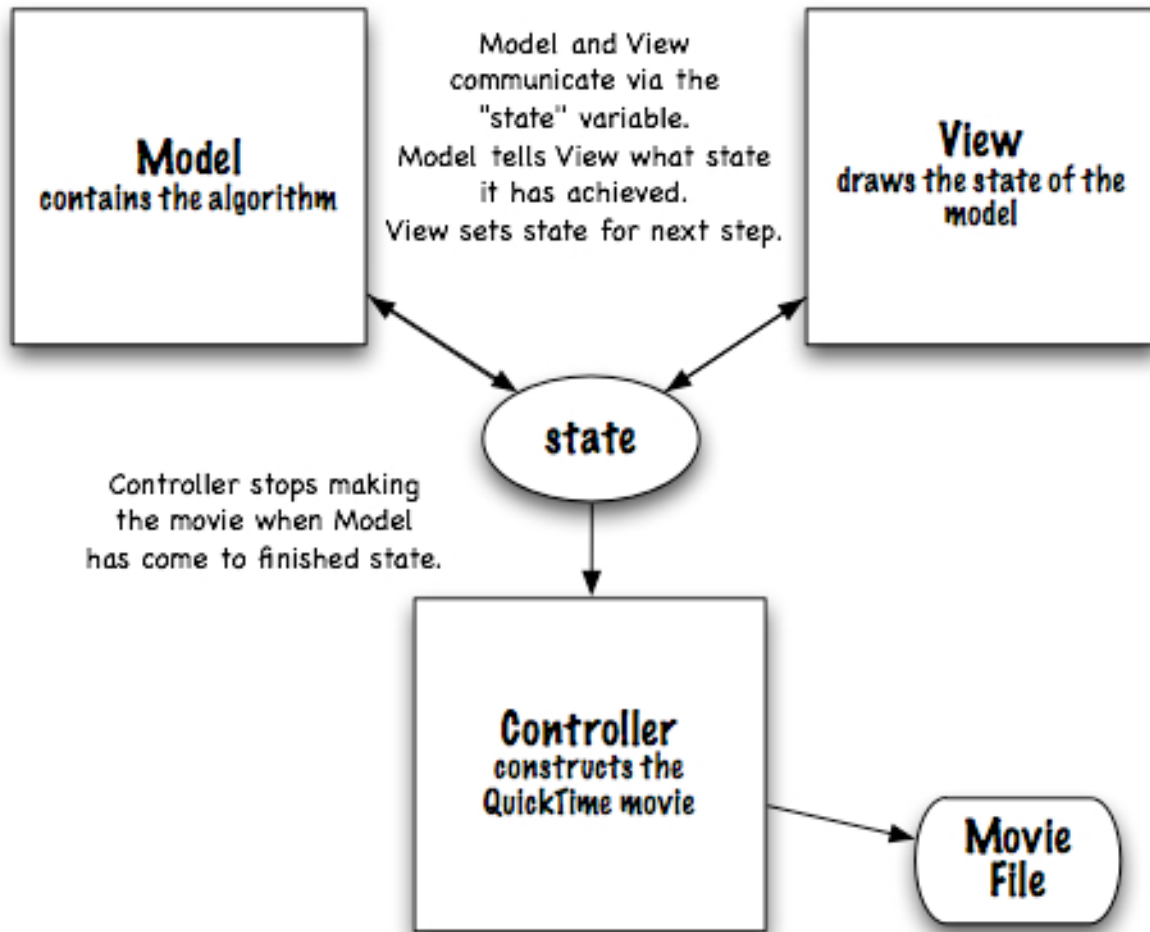


Figure 1. Model-View-Controller Paradigm.

Recently, we have implemented movie-making programs using Objective-C on the Mac OS X platform, with Xcode as the IDE. The development work has focussed exclusively on creating the video track and one or more text tracks to illustrate and explain each algorithm. We also use third-party software to provide headers, trailers and breakpoints.

Workflow. As an example, we present a movie of the drawing of the Lorenz attractor^{15 16} for a particular combination of parameters and initial conditions. For this movie, the *model* is the set of three coupled nonlinear differential equations proposed by Lorenz, the *view* is the projection, scaling and coloring of the drawing provided by OpenGL primitives, and the *controller* passes each drawing to QuickTime and opens and closes files.

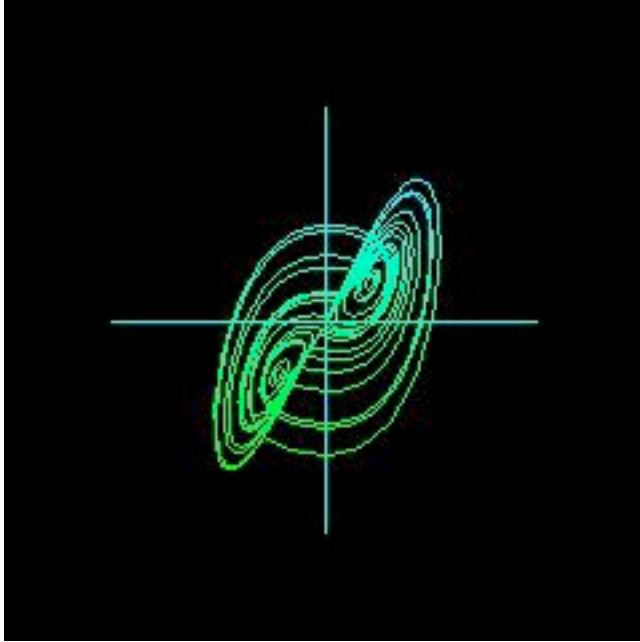


Figure 2. One frame of the QuickTime movie of Lorenz attractor

A QuickTime movie is a collection of images sequenced by time. This movie is made by photographing¹⁷ the Lorenz attractor curve as it grows through time. At each point in the development of the curve, a photograph is taken and passed to a CODEC for incorporation in video media and track by QuickTime; in addition, a sentence or phrase of text is stored in a buffer and passed to a text handler for incorporation in text media and track by QuickTime. At the end of the movie-making, the two tracks are integrated into a movie which is written to a file.

A QuickTime Virtual Reality (QTVR) movie is a collection of images sequenced by spatial position. To make a QTVR movie, the camera can be moved around the scene, or the scene can be rotated in front of the object. Our QTVR movies have been made by taking 450 images, every 12° of latitude (84° south to 84° north) and 12° of longitude (0° to 360°). The user can grab and manipulate the curve in three-dimensional space.

Both the QuickTime and QuickTime virtual reality movies are made with the same program using the same Model, View and Controller. There are three states (three scenes in the play): the growth of the Lorenz curve; moving around the Lorenz curve (as part of the time-based sequence); and moving around the Lorenz curve in both latitude and longitude to form images for the QTVR movie. When enough segments of the curve have been drawn, the state variable advances; when the camera has moved once around the scene, the state advances; and when 450 images have been photographed, the state advances (to the "end-of-movie" state).

Additional features. The QuickTime movies have one or more text tracks containing annotation in one or more languages. In addition, we add headers and trailers to each movie to provide a title, instructions and credits. The headers and trailers are constructed as graphics and placed into a picture track before and after the video and text tracks.

We construct a question or comment to the student or observer as a graphic, and placed into a picture track at the time at which the movie should pause and the student consider the question. The picture is made with a transparent background, so that the student can see the geometric construction through the question. Figure 3 shows a typical question overlaying a frame of a movie; this figure is taken from Fortune's algorithm for constructing the Voronoi diagram. An invisible "sprite" is constructed whose only purpose is to stop the movie at the time the question should be read by the student; the sprite is placed in a sprite track and synchronized with the appearance of the question. Figure 4 shows alignment of sprites in the track "pauses," pictures in the track "Questions," headers, text, and video in a five-track movie.

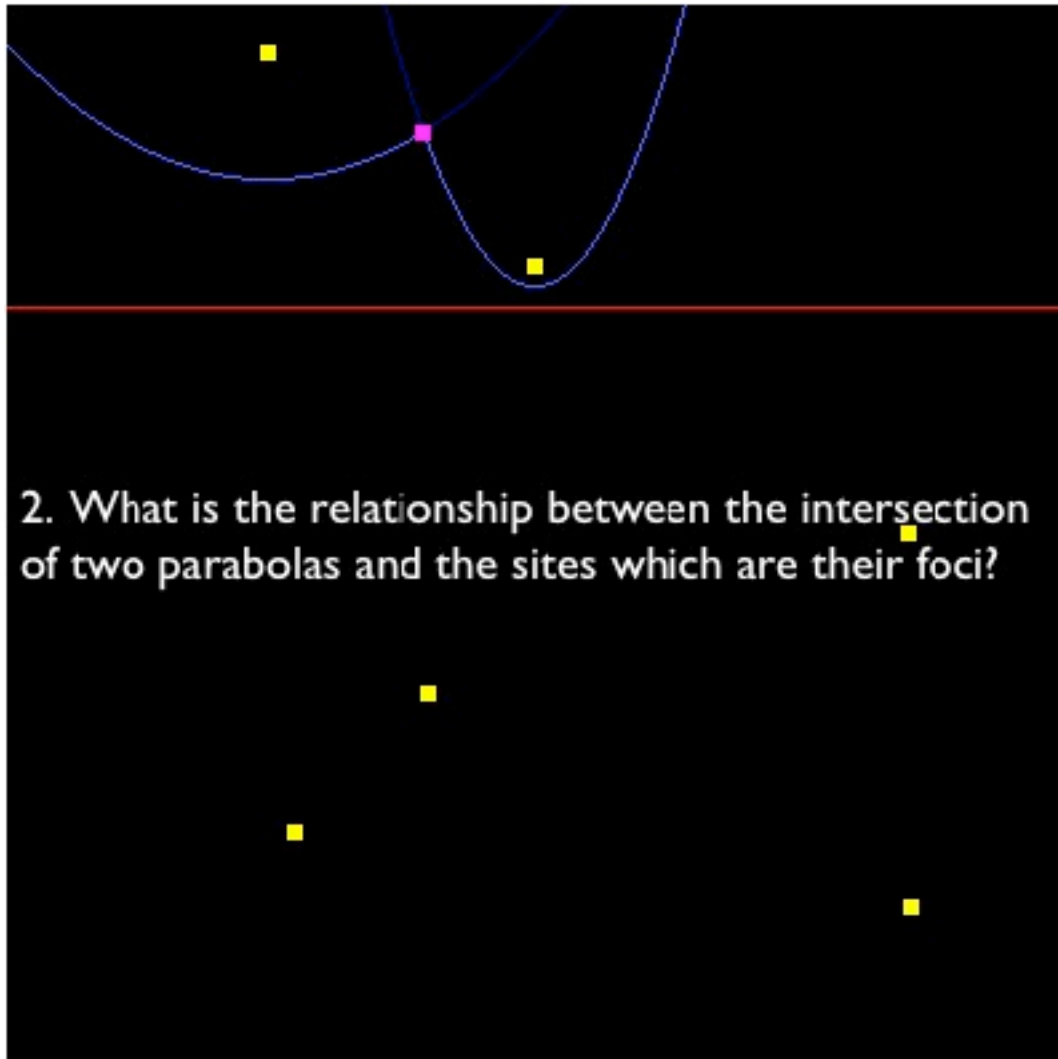


Figure 3. Question Superimposed on Video Track:from Fortune's Algorithm for Constructing the Voronoi Diagram

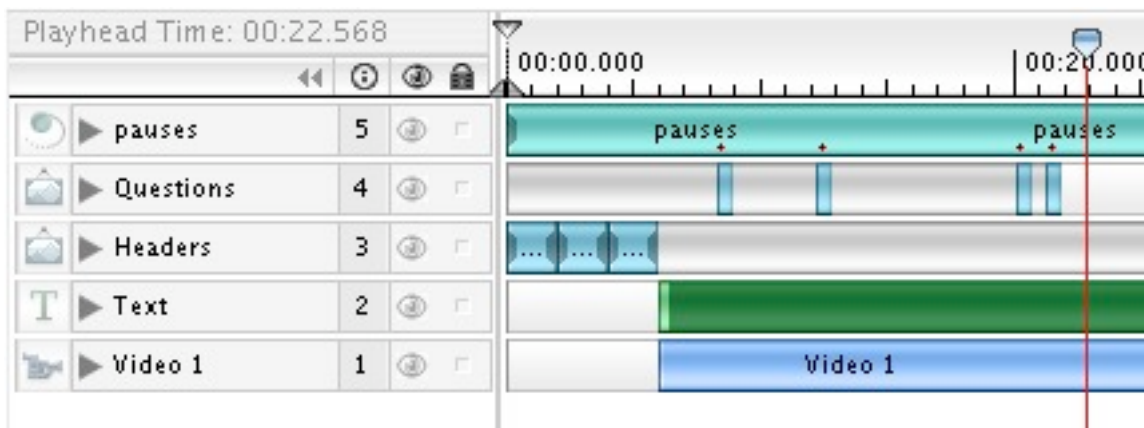


Figure 4. Alignment of Sprite (pauses), Picture (Questions), Picture (Headers), Text and Video in a Composite Movie

Additional Software. We have used *The VR Worx* (VR Toolbox, Pittsburgh, PA) to stitch together images to make a QTVR movie. We have used *Omnigraffle* (The Omni Group, Seattle, WA) to construct the graphics which become headers, trailers and questions. Finally, we have used *LiveStage Pro* (Totally Hip Software, Vancouver, BC, Canada) to combine picture, sprite, video and text tracks. The question overlay in Figure 4 was created with *Omnigraffle*, and the alignment shown in Figure 5 is taken from *LiveStage Pro*.

The Product

The following table shows the movies we have created to visualize algorithms from twelve problems in computational geometry. This suite of QuickTime and QuickTime Virtual Reality movies consumes over 2.5 Gb of disk space, and is available from the author's website or on CDs or DVDs from the author.

PROBLEM	ALGORITHM	CONDITIONS
Binary space partitioning tree		Forming and querying a BSP tree
Convex hull (2D)	extreme point	8, 16 points
Convex hull (2D)	extreme edge	8, 16, 32 points
Convex hull (2D)	"Gift Wrap"	25, 50, 100 points
Convex hull (2D)	"QuickHull"	10, 25, 50, 100 points
Convex hull (2D)	incremental	20, 50, 100, 200 points; bi- and trilingual
Convex hull (3D)	incremental	7 points; QTVR added
Delaunay triangulation	from the Voronoi diagram	6, 10, 16 points
Delaunay triangulation	incremental	6, 10, 16 points
Line intersection	"brute force"	50, 100 line segments
Line intersection	sweepline	50, 100, 200, 500 line segments
Motion planning		path-finding; shortest path
Point-in-Polygon	plumbline	on a map of Wicomico Co., MD, USA

PROBLEM	ALGORITHM	CONDITIONS
Point-in-Polygon	trapezoidal map	on a map of Wicomico Co., MD, USA
Point-in-Polygon	winding number	on a map of Wicomico Co., MD, USA
Polygon triangulation	recursive	8, 16, 32, 64 sides
Polygon triangulation	"Art Gallery"	8, 16, 32 sides
Polygon triangulation	make monotone polygons	8, 16, 32, 64 sides
Polygon triangulation	triangulate monotone polygons	8, 16, 32, 64 sides
Space partitioning	painter's	3, 5 objects
Search tree (1D)	construct and search	8, 16, 32, 64 points
Search tree (2D)	construct	8, 16, 32 points
Search (range) tree (2D)	construct	8, 16 points
Voronoi diagram	intersection of half-planes	6, 10, 16 points
Voronoi diagram	Fortune's	6, 7, 8, 10 points
Voronoi diagram	quadtrees	6, 10 points; 4 and 1 pixel resolution

Table 1. The Suite of Visualizations of Algorithms from Computational Geometry

Evaluation and Future Directions

Evaluation. It has been our philosophy that one cannot evaluate the effectiveness of algorithm visualization in computational geometry until one has enough visualizations to use. We hope that we have created a sufficiently rich suite of algorithm visualizations that instructors now have an adequate resource, all with similar formats and operating instructions. We hope to receive comments on the utility and effectiveness of these visualizations, as well as on their design and technical accuracy.

Many of the visualizations have been constructed by students. Implementing a visualization of an algorithm requires a careful understanding of the algorithm itself, and it is no surprise that students repeatedly remark that they only understood the algorithm after they both implemented it and constructed an animation of it. We also hope that providing a very simple Model, View and Controller in Objective-C will enable others to implement visualizations of algorithms, thereby gaining both experience and understanding.

Questions for the future. In both references used for computational geometry (deBerg, *et al.*; O'Rourke; notes 3, 4) there are additional problems, algorithms and datasets which could be visualized. We have done some work in animating algorithms from computational mathematics, and have seen the application of the QuickTime movie to other areas in mathematics and computer science. We have only just begun to explore the use of QuickTime Virtual Reality movies in algorithm visualization, and to construct ways to deliver combined QuickTime and QuickTime VR movies. Is the QTVR movie useful?

Other than to provide a convenient container for a composite QuickTime and QuickTime VR movie, is a branded or "skinned" player a useful adjunct to the movies? It is now possible to deliver QuickTime movies for playback on hand-held devices; is this a useful platform for these or other visualizations?

Acknowledgments

The author gratefully acknowledges advice from both Apple Computer and Totally Hip, Inc., as well as the numerous students (note 2) who have contributed to the project.

¹ QuickTime is a software technology of Apple, Inc. for containing, managing, and displaying time-dependent media. See: <http://www.apple.com/quicktime/mac.html> .

² The plural pronoun is meant to include, along with the author, these students who have contributed one or more visualizations to the suite: Nicholas Bergson-Shilcock (2005), Shubhomoy Biswas (2003), Kevin Camasi (2001), Jeffrey French (1994), Lindsay Hilbert (2003), Donald McElheny (2005), Anne Peacock (2001-2), Daniela Schroll (*Universität Wien*, 2000), Linda Malick (2003), Shawn Simon (2000) and Anton Weber (*Universität Innsbruck*, 2003).

³ deBerg, M., *et al*, *Computational Geometry: Algorithms and Applications*. (Berlin, Heidelberg, New York: Springer-Verlag, 2000).

⁴ O'Rourke, J., *Computational Geometry in C*. (Cambridge, Melbourne, New York: Cambridge University Press, 1994).

⁵ Anderson, J. M., "Algorithm Animation using QuickTime Movies for Student Interaction: Algorithms from Computational Geometry. In *Proceedings of ITICSE*, Helsinki, Finland, (July 2000): 185.

⁶ Brown, Marc, *Algorithm Animation*. (Cambridge, Mass.: MIT Press, 1988)

⁷ Stasko, J. *et al.*, eds., *Software Visualization: Programming as a Multimedia Experience*. (Cambridge, Mass.: MIT Press, 1998).

⁸ See, for example, Rößling, G., Naps, T., *et al.*, "Merging Interactive Visualizations with Hypertextbooks and Course Management," *inroads* (Bulletin of ACM SIGCSE), 38 no. 4 (December 2006): 166-181.

⁹ ACM Symposium on Computational Geometry: Video Reviews. (ACM, 2003-2006; accessed 20 December 2006); <http://compgeom.poly.edu/acmvideos> .

¹⁰ Dreshan, H., *et al.*, "CCAA: The Complete Collection of Algorithm Animations," (Hope College, 2001; accessed 20 December 2006); <http://www.cs.hope.edu/algoanim/ccaa> .

¹¹ Stasko, J., *et al.*, "XTango." (Graphics, Visualization and Usability Center, Georgia Institute of Technology, 2001; accessed 20 December 2006); <http://www-static.cc.gatech.edu/gvu/softviz/algoanim/xtango.html> .

¹² Rößling, G., "Animal-Farm: An Extensible Framework for Algorithm Animation," in *Proceedings of the Second Program Visualization Workshop*, Aarhus, Denmark (University of Aarhus, Denmark (2002): 52-58).

¹³ Anderson, J. and Naps, T., "A Context for the Assessment of Algorithm Visualization Systems as Pedagogical Tools," *Proceedings of the First International Program Visualization Workshop*, Porvoo, Finland (University of Joensuu Press (2001): 121-130.)

¹⁴ The model-view-controller paradigm (MVC) was introduced in 1979 by Trygve Reenskaug and described in detail in Burbeck, S., "Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC). (ParcPlace Systems, 1987, 1992); available from <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html> (1997; accessed 20 December 2006).

¹⁵ Lorenz, E., "Deterministic Nonperiodic Flow," *Journal of Atmospheric Science* 20, (1963): 130-141.

¹⁶ Stewart. I., "The Lorenz Attractor Exists," *Nature* 406 (2000): 948-949.

¹⁷ We use the word "photograph" here to mean that images are formed using a virtual camera controlled by OpenGL functions.