Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

ADDING ANIMATION TO A SIMULATION USING PROOF™

James O. Henriksen

Wolverine Software Corporation
2111 Eisenhower Avenue, Suite 404
Alexandria, VA 22314-4679, U.S.A.

ABSTRACT

Proof Animation™ is a family of products for adding animation to discrete event simulations. Proof is available in a variety of versions, including an inexpensive, student version, mid-size and unlimited-size commercial versions, a run-time version, and a royalty-free, redistributable demo viewer. Proof is an ASCII-stream-driven, general-purpose animation system which runs on readily available PC hardware. Its vector-based geometry provides a large animation canvas and the ability to zoom in or out, while maintaining crisp, clear images. Proof includes built-in drawing tools and CAD import/export for ease in creating animation layouts. Proof's open architecture makes it ideally suited for serving as a concurrent or post-processed animation engine for models written in a wide variety of simulation and programming languages. Proof's superior power and performance assure smooth, realistic motion for animations, regardless of their size, complexity, or application. Proof uses Microsoft's DirectDraw™ interface for accessing video hardware. DirectDraw is a built-in component of Windows 98, NT4, and 2000, and it is available as an add-on for Windows 95. Proof is able to exploit high-performance MMX hardware.

1 INTRODUCTION

Proof Animation is a general-purpose animator designed for use with the widest possible variety of simulation tools. Every Proof animation requires two ASCII input streams, (1) a layout stream, describing static characteristics of an animation, e.g., the background drawing over which objects move, and (2) a trace stream, which is a time-ordered sequence of commands which create, destroy, move, and otherwise change objects displayed on the layout, portraying events in a simulation. Both of these streams are free-format ASCII text, with well documented ("open") architectures which can be generated easily in a variety of ways.

Proof can be used in post-processing mode or directly driven by another program. When Proof is used in post-processing mode, its input streams must be stored in files. Trace files are almost always written using a simulation language or package, such as SLX (Henriksen 1998), GPSS/H™ (Crain 1997) , Extend™, Slam™, Siman™, Simscript II.5™, etc. Trace files can be written using non-simulation languages, such as C, and simple trace files can even be prepared using a text editor. Layout files are almost always developed using Proof's built-in drawing tools. Proof also includes a CAD import feature, allowing quick importing of .DXF files. Layout files can also be generated by a program. While this is not done very often, it can and has been done straightforwardly.

When Proof is *directly* coupled to simulation software, input streams are transmitted to Proof one line at a time via subroutine call. Proof can be directly driven by any program which is capable of constructing C-compatible Dynamic Link Library (DLL) calls; i.e., the directly driven version of Proof is packaged as a Windows DLL.

Proof's open, simulation-language-independent architecture is a great strength, both technically and comercially. From the user's perspective, Proof provides the opportunity to add high-quality animation to a simulation developed using existing tools, with no requirement to purchase and use simulation tools provided by Wolverine Software. From Wolverine's perspective, Proof provides a stream of sales to users already committed to buying and using simulation tools provided by Wolverine's competitors.

Because language independence is such an important strength of Proof, every effort will be made to ensure that improvements to Proof preserve this independence. However, from time to time, improvements may be made to Proof solely for exploitation by Wolverine Software's simulation products, e.g., SLX and GPSS/H.

2 THE PROOF ANIMATION FAMILY

The Proof Animation product family runs on readily available, inexpensive PC hardware. All versions require a 486 or better CPU, a math coprocessor, and a VGA-compatible video card. Proof is able to exploit MMX (multi-media extensions) hardware, but MMX hardware is not required for running Proof. For non-MMX hardware, MMX features are emulated, incurring an overhead of 10-30%.

Proof is a Windows 95/98/NT4/2000 application. Future improvements to Proof will be made only to the Windows version of Proof. An older version of Proof which runs under DOS, Windows 3.x, and OS/2 as a 32-bit extended DOS application has been frozen. Running Proof as a Windows application requires DirectDraw driver support for the video hardware used. DirectDraw is an integral component of Windows 98, Windows NT4 (Service Pack 4 required) and Windows 2000. Direct-Draw run-time support is available free-of-charge as an add-on to Windows 95. A set of Windows icons is supplied with each of the Proof Animation products to provide single-click launching. The following products comprise the Proof Animation family:

- PROOF ANIMATION
  Proof Animation is an entry-level version of Proof. Memory size is fixed and limited. It includes built-in CAD import/export feature. It is adequate for small to mid-sized animations.
- PROOF PROFESSIONAL
  Proof Professional exploits all available virtual memory for animating large systems. It includes a built-in CAD import/export feature.
- RUN-TIME PROOF PROFESSIONAL
  Run-time Proof Professional runs developed animations or presentations, but has no animation *development* capabilities. It provides a low cost way to run different scenarios with a fixed layout file prepared using Proof Professional or Proof Animation.
- STUDENT PROOF ANIMATION
  The student version of Proof Animation is included with the *Using Proof Animation* text. Size and playing time limitations are imposed; otherwise it is identical to Proof Animation.
- PROOF ANIMATION DEMO MAKER
  Demo versions of animations can be prepared under a licensed copy of Proof Animation or Proof Professional containing the Demo-Maker add-on. Copies of the executable demo files can be reproduced and distributed free of charge and viewed by anyone.
- PROOF ANIMATION DEMO VIEWER
  The Proof Demo Viewer, which is available free-of-charge, is used to *view* demos prepared with the Demo Maker.

3 THE GENERAL-PURPOSE APPROACH

3.1 Loosely-Coupled Interface

While built to work easily with Wolverine's SLX and GPSS/H simulation software, Proof Animation also provides affordable and powerful animation software to users who develop models in other simulation and programming languages. This, in fact, was a major goal in the design of Proof. We wanted to develop an interface which made Proof easy to use as a "back end" for the widest possible variety of simulation-based "front ends." This approach allows prospective purchasers of simulation and animation software to adopt a mix-and-match strategy, basing their purchasing decisions on what they feel to be an appropriate combination of functionality and pricing. The number of success stories using Proof Animation with other software continues to grow.

3.1.1 ASCII Input Streams

Proof Animation is driven by ASCII streams. Therefore, any software capable of formatting ASCII text can be used with Proof Animation. Proof requires two ASCII input streams, a layout stream and a trace stream. The layout stream describes the geometric details of the background over which objects move, provides geometric definitions and properties for such objects, and defines logical paths along which the objects move.

Ordinarily, layout streams are produced at least in part by using Proof Animation's drawing tools; however, the layout stream command set specifications are published so programs can easily be written to generate layout streams. For example, some users have written front ends for their simulation models that allow different system design parameters to be specified for each run. Based on these parameters, different geometric configurations are written and incorporated into a layout stream. The new layout appears on screen when Proof Animation is invoked.

The trace stream contains a time-ordered sequence of commands such as CREATE, DESTROY, PLACE, PLOT, MOVE, SET SPEED, SET COLOR and many more. This stream provides Proof Animation with information on when, where, and what to create, destroy, place, plot, etc. Trace streams are free-format, and the commands are easily learned and used. They provide exactly the kind of flexibility necessary to easily be integrated with the simulation model logic. Any language that can produce formatted ASCII output can write a trace stream.

3.1.2 The Post-Processor Version of Proof

Proof is most commonly used as a post-processor, i.e., Proof runs *after* a simulation has run to completion. In post-processing mode, both the layout and trace streams must exist as *files* before invoking Proof Animation.

Two great advantages result from the post-processing approach. First, PC hardware is not shared between the simulation and the animation. This leaves the entire CPU for running the animation. Second, it provides the abilities to jump back and forth in time during the animation playback, to speed up or slow down the viewing speed, or show all or a specific portion of an animation. These features make it easy to investigate unusual system behavior or highlight points of interest.

3.1.3 The Concurrent (DLL) Version of Proof

A user-*callable* version of Proof is available in the form of a Dynamic Link Library (DLL). Any software which can format ASCII text commands and generate C-compatible subroutine calls can exploit the DLL version of Proof. Since trace stream information is generated "on-the-fly," concurrent, or even real-time, animation is possible with the DLL version of Proof.

The Proof DLL provides the following functions:

Function	Purpose
ProofDLL	Initialization/shutdown
ProofOpenLayout	Opens a layout file
ProofSendTraceLine	Sends a trace command
ProofSuspend	Temporarily suspends Proof
ProofResume	Resumes animation
ProofStatus	Retrieves status of Proof

The DLL performs all required synchronization between Proof and the software sending commands. Proof maintains a small queue of commands, helping to iron out the peaks and valleys in its own demands for computing time and those of the driving application. Proof yields time to the driving application that would otherwise go wasted, allowing the driving application to get ahead of Proof or to refill the command queue if the driving application has fallen behind.

4 GEOMETRY, MOTION, COLOR, AND RESOLUTION

4.1 Vector-Based Geometry

In the Proof Animation product family, all layout and trace information is based on vector geometry. Vector-based descriptions are automatically mapped into pixels to build a screen image. One of the advantages of this approach is that layouts can be much larger than a single screen. With the ability to zoom in or out and pan, larger layouts are easily navigated to show the overall layout or zoomed in to whatever level of detail is necessary. Vector-based geometry also provides the ability to have moving objects realistically *rotate* around corners instead of the sliding effect to which other animation packages are limited.

Another advantage of vector-based geometry is that most CAD packages are capable of producing standard vector-based .DXF files. In many cases, a CAD drawing already exists for the system to be animated. If so, the effort of redrawing an entire layout can be avoided. Proof Animation's built-in CAD Import/Export feature provides the capability to convert industry-standard .DXF files into Proof Animation layout files, *and vice versa*. Credibility of the study is enhanced when viewers see a familiar CAD drawing of the system integrated into the animation. These advantages maximize the power of the animation by giving a user total flexibility on the detail and complexity of the drawing.

4.2 Smooth Motion

Proof Animation is able to achieve very smooth motion for large numbers of objects. At all times, Proof maintains two images, the visible screen image and a hidden image. All updating is done using the hidden image. When an update is completed, the hidden and visible images are flipped instantaneously. Flipping is triggered by the video hardware's vertical retrace interrupt, which is generated each time the electron beam painting the screen reaches the lower right corner of the screen. Vertical retraces occur at a rate of 60-70 times per second. By updating the screen at the hardware's native screen refresh rate, Proof is able to achieve very smooth motion of hundreds of objects. Other software can often sustain refresh rates of only 5-10 updates per second. The ultimate purpose of an animation is to achieve a realistic depiction of the system being studied, allowing the audience to gain confidence in the results of the simulation study. Objects that move smoothly across the screen are more realistic than those that jump across the screen.

4.3 Color and Resolution Options

All versions of Proof, including the student version, provide for operation in 256-color mode in a variety of screen resolutions. Proof supports 640x400, 640x480, 800x600, 1024x768, and 1280x1024 screen resolutions. Higher resolutions are available only if sufficient video memory exists for storing at least two full screen images. For example, 1024x768 resolution cannot be used on hardware which has only 1MB of video memory, since at least 1.5MB is required to store two screen images. We use the phrase "at least," because the ways in which video driver software manages video memory may in some instances impose additional overhead, requiring more video memory than would normally be expected. 1024x768 resolution is now commonplace on desktop computers. Most new laptop computers support at least 800x600 resolution.

Two groups of 32 colors are available for end-user use, one group for foreground colors and one group for background colors. When two or more objects or layout elements overlap, the color which appears on the screen is determined by making a color comparison. Higher-numbered colors take precedence over lower-numbered colors. Backdrop colors have lower color numbers than foreground colors, so foreground objects will always move over backdrop-colored objects. If two foreground-colored objects overlap, the higher color numbers are made visible. The use of color comparisons makes it possible to easily animate such things as freeway overpasses and underpasses. Color comparisons are performed by using MMX instructions, which can compare eight 8-bit pixels simultaneously.

The remaining colors are dedicated for use by Proof and Windows. Windows reserves 20 colors for its own use.

5 CREATING ANIMATIONS AND PRESENTATIONS

5.1 Drawing the Layout

The first step in developing an animation is to draw a layout. If a CAD drawing of the system is available in the form of a DXF file, a user can begin by importing the drawing into Proof, using Proof's built-in CAD import/ export utility. Once imported, the drawing can be examined by layer or by line style. Large CAD drawings typically contain layers and line styles which are inappropriate for inclusion in an animation layout. For example, one or more layers may be used for displaying dimensions or annotations. Such layers can be quickly eliminated, as can line styles used for unwanted hidden lines and center lines. The resultant drawing is saved as a Proof Animation layout file. The original .DXF file remains intact.

If a user does not have a CAD drawing or prefers to draw using a computer, (s)he can use the drawing tools provided in Proof's Draw Mode. Although it is mouse-oriented, Draw Mode also allows keyboard input, so if a user needs to draw a line of a specific length at an exact angle, (s)he can enter these specifications *numerically.* To help in drawing scaled, accurate layouts, a visible grid is turned on automatically when Draw Mode is entered. For additional aid in drawing, Proof has a Snap-to-Grid option. This option is also *on* as the default setting. Snap-to-Grid limits the drawing of layout elements from grid point to grid point, thus eliminating the chance of small gaps between the endpoints of *seemingly* connected lines. Other snap options which help draw accurate layouts are Snap-to-Endpoint which *magnetically* attracts the mouse cursor to the ends of lines and arcs, and Snap-to-Tangent which quickly finds points of tangency between lines and arcs. All of these options can be turned on or off by the user during the drawing session.

5.2 Defining Object Classes

Once the background of the animation is drawn, the second step in developing an animation is to define one or more object classes. This is done in Class Mode. Objects and object classes are among the most important constructs in Proof Animation. A class provides the geometric description of the individual objects that move throughout the animation. The class definition also includes the initial properties such as physical clearances, color, and speed of the individual objects. Each animation will usually have a collection of object classes.

It is helpful to think of an object class as the template from which the individual objects are made. An individual object is based on the single geometric description of a particular object class. There can be an arbitrary number of *objects*, such as widgets, in the system at once, but there need be only *one* widget object class.

Motion and color-changing commands in the trace stream operate on *objects*. The drawn background components, produced in Draw Mode, cannot be moved or changed. If dynamic changes in background elements are required, the appropriate components must first be defined as object classes and then created and positioned directly in Draw Mode. Objects that are created and placed in the layout while the user is drawing the background are called *layout objects*. Layout objects enable a user to scale and position the objects into the layout while having the background components visible as reference points. While the animation is running, layout objects can be manipulated using trace stream commands. For example, if an idle machine is shown as green and a busy machine as red, the machine must first be defined as an object class. Objects from that class can be created and placed as part of the layout stream, and their color can be changed while the animation is executing.

5.3 Defining Paths

Proof Animation provides two kinds of motion: absolute and guided. Absolute motion, specified by the MOVE trace stream command, causes an object to be moved between two points. Guided motion always occurs along a fixed route, called a path. For guided motion, such as travel on conveyors or along guide wires, the next step in the animation development is to use Path Mode to define one or more paths.

Paths are comprised of lines and arcs that represent the route that the objects will follow. This underlying geometry must first be drawn using Draw Mode or be imported from a CAD drawing. The logical path segments are then defined as a logical superstructure imposed on top of existing lines and arcs. A single line or arc can be part of one or more paths. Once defined, paths are saved as part of a layout file.

Using paths is very simple because Proof Animation does all the work. The most commonly used trace stream path command is PLACE *object*ID ON *path.* Once an object is placed on a path, it will follow that path until it comes to rest at the end of the path or until it is PLACEd elsewhere or DESTROYed. All objects traveling on the same path can be stopped simultaneously and resume movement at a later time. Paths provide outstanding animation power in response to a single trace stream command.

Accumulating paths provide even greater power for animating paths on which queuing can take place. On accumulating paths, Proof Animation reflects physical reality by *visually* queuing objects when bottlenecks occur. This often makes a simulation model of the system much simpler to construct, because such queuing need not always be explicitly represented in the model code. Most systems contain some accumulation. This property can be used to represent certain types of conveyors, cars at a traffic signal, bank lines, and more. Paths play an especially important part in transportation, product flow, and material handling animations.

5.4 Writing the Trace Stream

The next step in the animation development is producing the animation trace stream. Trace streams consist of very readable ASCII commands. Trace streams are time ordered. Groups of one or more animation events take place instantaneously between TIME commands. Consider the following portion of a trace stream:

```
TIME  34.6
CREATE  PLANE 1
PLACE 1 ON RUNWAY3
SET  1 SPEED 75
TIME  52.8
```

It is very easy to visualize the results of these commands. At time 34.6, an object with an ID number of 1 is created with geometry and properties inherited from class PLANE. This object will appear on screen at the beginning of a path named RUNWAY3 and begin moving along the path. The speed at which object 1 will move is set to 75 units of distance per unit of simulated time. These units are user-determined, e.g., feet and seconds. Proof will continue reading trace stream commands until it reads the TIME 52.8 command, signaling the end of the events that begin at time 34.6. At this point, processing of trace stream commands is suspended until time 52.8 is reached. The ratio of simulated time to viewing time is constant, but user-specified.

It is very easy to produce simple trace streams such as the one shown above with any ASCII editor. However, for most applications, it is impractical to create trace streams by hand. Using a simulation model or program to generate the trace stream is usually the *only* viable approach. In order to produce a trace stream, output statements are inserted into the simulation model to write the appropriately formatted commands. One should think of this process as building a model of a model. Just as a model omits certain details of the system it represents, an animation omits many details present in a model. One must decide exactly which details are important enough to warrant their inclusion in an animation. For each event in the model which needs to be portrayed on the screen, one or more trace stream commands must be generated. Typically, the number of points at which trace stream commands must be generated is quite small. For example, an animation of a small, but complicated prototype conveyor system shown at the 1997 Winter Simulation Conference required the introduction of only 14 statements into the simulation model to produce a high-quality animation.

The Proof Animation trace stream command set has been designed to be easily generated. Any language with the ability to write a formatted ASCII stream is capable of producing a trace stream.

5.5 Building a Presentation

As an optional final step, one can construct a high-quality presentation comprised of snippets of animation, slides, text, and sound. Segments of a presentation can be linked together using fades, dissolves, and other special effects to add polish. Presentations are defined by preparing simple ASCII presentation script (.PSF) files. As of this writing, one must use a text editor to prepare a .PSF file; however, a built-in presentation editor is contemplated for future versions of Proof.

Most presentations are implemented using grouping and subgrouping constructs which allow viewers of the presentation to navigate their way through the presentation *hierarchically*. Navigation is accomplished by means of a Windows "Tree View," as shown in Figure 1. Within a Tree View, subgroups are prefixed with small boxes containing a "+" or "-". Clicking on a "+" box causes a compressed subgroup to be expanded and changes the "+" to a "-". Clicking on a "-" compresses a displayed subgroup into a single line.

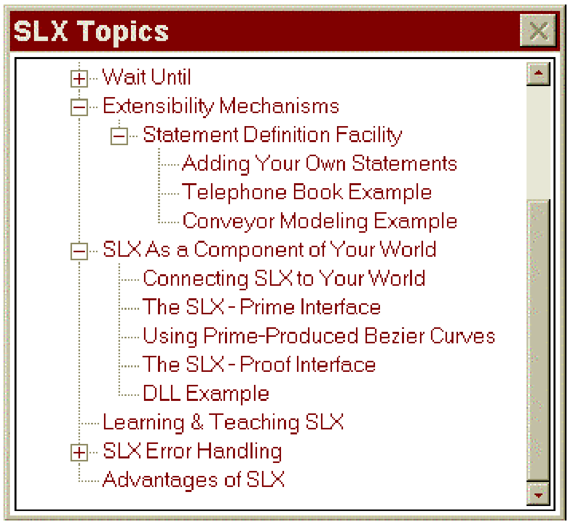
SLX Topics
- ⊞ Wait Until
- ⊟ Extensibility Mechanisms
  - ⊟ Statement Definition Facility
    - Adding Your Own Statements
    - Telephone Book Example
    - Conveyor Modeling Example
- ⊟ SLX As a Component of Your World
  - Connecting SLX to Your World
  - The SLX - Prime Interface
  - Using Prime-Produced Bezier Curves
  - The SLX - Proof Interface
  - DLL Example
- Learning & Teaching SLX
- ⊞ SLX Error Handling
- Advantages of SLX

Figure 1: Hierarchical Presentation Navigation

Slides used in a presentation can be .BMP files, .PCX files, or .RTF files. RTF files can be prepared using tools such as Microsoft WordPad. Using WordPad, one can quickly prepare textual slides which incorporate color and a variety of typefaces. Slides can be also be created by using Proof's built-in screen-grabber or by using any software package capable of exporting industry-standard .BMP or .PCX bitmap files. There are many such packages available, and virtually all of them can produce very high-quality charts, graphs, and slides.

Presentations can be developed so that slides and animations appear on the screen for a defined amount of time. The viewer does not have to interact with the computer for the presentation to continue. Presentations can also be developed to continue once a key or mouse button is pressed, giving the viewer or presenter ample time to comment on what is currently on the screen.

When developing the presentation, a user can choose to highlight areas of interest within the animation by using different views or sounds (.WAV files) to draw the viewer's attention to particular aspects of the animation.

Proof is a well-behaved Windows application. During the course of a presentation, one can switch out of and back into Proof, if necessary, to run other applications. Automatic transitions to/from other applications can be incorporated into a presentation by using the "syscall" presentation script command.

6 EXTEND + PROOF

The DLL version of Proof has been used to build animation extensions to Imagine That's Extend version 4 and version 5 simulation software. For Extend version 4, only "off-the-shelf" Proof features were used. No custom modifications of Proof were required. Imagine That has cleverly exploited the architecture of Proof to offer a Proof interface for Extend users that is very user-friendly. For example, Extend is able to read Proof layout files and extract the names of paths and object classes defined in the layouts. This is possible because (1) the syntax of the layout file is openly documented, and (2) the format of a layout file is simple enough to be easily read. When an Extend user wishes to place an object onto a Proof path, (s)he is presented with a menu of path names that have been read from a layout file. Thus, it is impossible for the Extend user to construct a Proof command that refers to a non-existent path.

The Extend version 4 user interface added a Proof block to Extend. In order to add animation to an Extend model, users simply inserted Proof blocks at points at which simulation events requiring animation actions occurred in a model. In Extend version 5, this interface has been improved. The dialogs for many Extend blocks now include an animation tab. When this tab is clicked, options for adding Proof commands automatically appear. Thus, with version 5 it's possible to add Proof commands to an Extend model without having to insert new blocks in a model. To further enhance the integration of Extend version 5 and Proof, a number of small, Extend-specific modifications were made to the standard Proof DLL, resulting in hand-in-glove integration of the two products.

7 SUMMARY

Wolverine Software's Proof Animation has set a standard for maximum power and performance. Proof Animation is not tied to a specific simulation language or application. Proof's features make it an ideal choice for the animation of systems such as manufacturing and material-flow applications, computer networks, health care applications, transportation, process reengineering, and many others, while maintaining ease of use.

An animation benefits a user in every phase of a simulation study: verification, validation, presentation of results, and the overall system design process. Proof Animation's unmatched features make it the perfect tool for each of these phases regardless of the application.

REFERENCES

Crain, R.C. 1997. Simulation using GPSS/H. In *Proceedings of the 1997 Winter Simulation Conference*, eds. S. Andradóttir, K. Healy, D. Withers, B. Nelson. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Earle, N.J. and Henriksen, J.O. 1994. Proof animation: reaching new heights in animation. *Proceedings of the 1994 Winter Simulation Conference*, eds. J. Tew, S. Manivannan, D. Sadowski, and A. Seila, 509-516 Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Henriksen, J.O. 1998. Stretching the Boundaries of Simulation Software. In *Proceedings of the 1998 Winter Simulation Conference*, eds. D. Madeiros, E. Watson, M. Manivannan, J. Carson. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Wolverine Software. 1995. *Using Proof Animation (Second Edition)*. Annandale, Virginia: Wolverine Software Corporation.

AUTHOR BIOGRAPHY

JAMES O. HENRIKSEN is the president of Wolverine Software Corporation. He was the chief developer of the first version of GPSS/H, of Proof Animation, and of SLX. He is a frequent contributor to the literature on simulation and has presented many papers at the Winter Simulation Conference. Mr. Henriksen has served as the Business Chair and General Chair of past Winter Simulation Conferences. He has also served on the Board of Directors of the conference as the ACM/SIGSIM representative. He can be reached via e-mail at: `<mail@wolverinesoftware.com>`.