Figure 4.30: A screen shot depicting the run phase of a Balsa session for a number sorting algorithm. The numbers are represented by vertical columns, the magnitude of each number is represented by the height of the corresponding column. As the numbers are sorted by the algorithm, the columns are moved and reordered by height.

contributes to the design of the graphical representations used. The animator’s task is then to implement the views that make up the graphical presentations. The scriptwriter is the person who constructs the scripts for the animation, i.e. what information is shown to the end user and when. Finally, the end user makes use of these scripts to view the dynamic graphical representations of a program’s algorithm.

The interaction style for the end user is referred to as a “set-up and run” cycle [Brown, 1988]. In the set-up phase the end user arranges the display layout, the algorithms they wish to view, and the parameters they want to associate with each algorithm (including its input generator and output views, see Figure 4.29). Once set up the end user runs the algorithm and observes the results (see Figure 4.30).

Balsa does not support the bi-directional control of the program’s execution. The user can either run the program and stop at the next stoppoint, pause at the next stoppoint, stop at the next steppoint, pause at the next steppoint or reset the program back to the start of the execution. The terms “stoppoint” and “steppoint” are taken from Mac Pascal; stoppoints are more commonly known as breakpoints, i.e. points inserted into the program to stop its execution, steppoints are equivalent to the steps of the program’s execution i.e. a steppoint occurs after every command.

Further information on Balsa can be found in [Brown and Sedgewick, 1985], [Brown, 1987], or
Algorithm Mapping Animation

\[
\begin{align*}
  x &= 10; \\
  \text{if } (y == 12) \\
  z &= 2.3; \\
  \text{for } (i=1; i<=10; ++i) \\
  a[1] &= 0.0;
\end{align*}
\]

Figure 4.31: John Stasko’s algorithm animation framework as used in TANGO. This figure was taken from [Stasko, 1989, page 34].

[Brown, 1988].

**Stasko - TANGO**

TANGO the Transition based ANimation GeneratiOn framework and system was developed by John Stasko while at Brown University. TANGO was devised for describing, specifying, analyzing and formalizing the elements involved in animating algorithms [Stasko, 1989]. The framework contains three primary components; namely: the “algorithm,” “mapping” and “animation” components (Figure 4.31).

The algorithm component adopts an event-driven approach in which any events important to the algorithm’s semantics are identified by the algorithm designer and are referred to as “algorithm operators.” These are then used to model procedure calls, mapping the algorithm to the animation. The procedure calls are then used to create the animation control file which constitutes the mapping component of the framework. The animation component contains the graphical objects, whose location, size and colour will change during animation, and the operations that control the animation. The approach devised here for generating smooth animations is referred to as the “Path Transition Paradigm” [Stasko, 1990].

Four abstract data types are used within the path transition paradigm; “images,” “locations,” “transitions” and “paths.” Images are either “primary images” such as lines, rectangles, circles and text, or “composite images” which are collections of primary images with specified geometric relationships. Locations are simply positions within the animation co-ordinate system, identified by
CHAPTER 4. REVIEW OF RELATED WORK

Figure 4.32: A screen view taken from a TANGO animation of a first-fit binpacking algorithm. The elements are inserted into the rectangle and tried against each column position until a large enough free-space is found to house them. The control bar shown at the bottom of the figure allows the user to pan around the view, zoom in and out, switch the debugger on/off, alter the refresh rate, and close the view.

Further information on TANGO can be found in [Stasko, 1989], [Stasko, 1990], or on the world wide web, see http://www.cc.gatech.edu/gvu/softviz/SoftViz.html

**Domingue, Eisenstadt and Price - Vital and Viz**

The Vital project was a four and a half year ESPRIT II research and development project, completed in April 1995, involving nine organizations in five different countries. The aim of the project was to provide both methodological and software support for the development of large, industrial, embedded Knowledge-Based System (“KBS”) applications. SV was seen as an opportunity to enhance the users’ control of the individual tools within the Vital Workbench. In order to support this a separate visualization framework and software library called “Viz” was created [Domingue et al., 1992]. Viz

---

Note the “Viz” visualization framework should not be confused with the “Vis” GA visualization tool developed by Annie Wu (see Section 4.2.1).
enables the user (i.e., KBS developer) to define and construct visualizations of their systems using a very high level programming language. A program’s execution data is stored in a history database which is used as the basis for creating different views of that program’s execution. These views are then made available to the user who can choose which views are displayed.

To orchestrate this Viz uses a story-telling metaphor in which the program’s elements (i.e., functions, data structures, lines of code, etc.) are referred to as “players.” The players are identified by the user and annotations are made either to the code or the code interpreter, such that the player’s values are recorded in the History database when interesting “events” occur. A diagram of the Viz architecture showing the different sub-components of Viz is given in Figure 4.33.

There are four main components to Viz, namely the “History,” “Views,” “Mappings,” and “Navigators” components. The history component holds a record of all key events that occur over the duration of the program’s execution. The views component provides the styles in which a particular set of players, states, or events can be presented. The mappings are the encodings used to present the players’ state changes, either graphically, or audibly within each view. Finally, the navigators are the tools or techniques used to interact with the user. They allow the user to traverse a view, move between multiple views, change scale, compress or expand objects, and move forwards or backwards through the program’s execution.

The Viz visualization framework and software library is capable of producing not only program visualizations (i.e., program data and code visualizations) but also algorithm animations. The extent to which the Viz framework and library is used within the Vital project is illustrated in Figure 4.34. The Problem Solving Architecture and Code Visualizations are examples of program visualizations,
they closely illustrate the actions of the code and states of the data being manipulated by the KBS. The Domain and Expert Scripted Visualizations are similar to algorithm visualizations where abstract representations are used to illustrate the KBS’s operations.

Further information on the Viz framework and the VITAL Workbench can be found in [Domingue et al., 1992], [Domingue et al., 1993] and on the world wide web, see

http://kmi.open.ac.uk/people/john/sv/viz/viz.html

http://kmi.open.ac.uk/people/john/vital/vital.html

**Brown and Najork - ZEUS**

After developing Balsa Marc Brown went to work at the Digital Equipment Corporation (DEC) where, along with Marc Najork, he developed an algorithm animation system called ZEUS. This was designed to provide support for both algorithm animation and multi-view editing.

The use of annotations to indicate “interesting events” in an algorithm is still used, however, added features include the use of objects, strong typing, parallelism and the graphical development of views.
Figure 4.35: A screen shot taken from a Zeus binpacking algorithm animation. A control panel is shown in the top right window, a code visualization is shown in the bottom right window, an algorithm animation is shown in the bottom left window, and the algorithm's progress is shown in the top left data window.

[Brown, 1991]. The use of objects encourages the reuse of code and facilitates the construction of composite views. The introduction of a graphical editor aids the construction of new view components and the adoption of strong typing provides an opportunity for generating automatic visualizations. A screen shot taken from a Zeus binpacking animation is given in Figure 4.35.

Further information on Zeus can be found in [Brown, 1991], [Brown and Hershberger, 1992], [Brown and Najork, 1993], or on the world wide web, see http://www.research.digital.com/SRC/zeus/home.html

**Stasko - Parade and Polka**

After John Stasko developed Tango he moved to the Georgia Institute of Technology where he created Parade, a PARallel program Animation Development Environment. The focus of the Parade project was to enable the use of “application-specific” visualization to assist the debugging and correctness-checking of parallel programs. Application-specific program views in this context are defined as views that illustrate the program’s semantics, its fundamental methodologies and the inherent application domain.
Parallel Program or all () {
    x = 13;
} fct2();

Animation Choreographer

Visualisation Paradigm - "POLKA"

Figure 4.36: An overview of PARADE highlighting its three major components; the "Parallel Program" component extracts the information required for producing the visualizations, the "Animation Choreographer" gathers the program information from the parallel program component and organizes it into a preferred format, and the "Visualization Paradigm" takes the choreographed program details and presents them in an apparently continuous smooth animation to the user. Any user interaction is passed to the animation choreographer by the visualization paradigm where it is acted upon. This figure was taken from [Stasko and Kramer, 1992, page 4].

PARADE is made up of three components; the "parallel program," "animation choreographer," and the "visualization paradigm" (Figure 4.36). The parallel program component extracts the necessary program information on which to base the views. The animation choreographer is responsible for the gathering of the program information and its subsequent organization into a preferred structure identified by the user (via the visualization paradigm). The third component, the visualization paradigm, passes the user's actions back to the animation choreographer and presents the choreographed program details in a smooth animated form. The visualization paradigm in PARADE is called POLKA (Parallel Object-oriented Low Key Animation) [Stasko and Kraemer, 1992]. POLKA is an object-oriented system written in C++ that provides high-level graphical-object primitives and motion primitives for the construction of algorithm visualizations and animations. POLKA is available for both the X Windows and Silicon Graphics GL systems; the Silicon Graphics GL version supports the use of 3D graphics.

The POLKA animation methodology is a combination of principles from the path transition paradigm [Stasko, 1990] and traditional 3D production animation systems. Figure 4.37 illustrates the hierarchy of a POLKA animation. An animation is made up of a series of Views with each view being made up of "Locations," "Actions" and "AnimObjects." An AnimObject is the base class for all graphical objects (either 2D or 3D); objects are created by the "Originate" method and deleted by the "Delete" method. Locations in POLKA can be used to reference and remember important positions.
Polka maintains the simple modification of graphical objects along paths approach cultivated in Stasko's path transition paradigm and adds the capability to program actions into objects at desired animation times. Two screen images illustrating both 2D and 3D visualizations from Polka are shown in Figure 4.38. The view on the left of the first image is a "blocks view" showing each element in an array as a block whose height indicates the element's value, and horizontal position indicates its position in the array. The view on the right is a "chart view" in which the horizontal lines are used to represent the swapping of elements; the start and end points of these lines indicate the positions of the elements being swapped. Colour is used in both views to indicate the partitioning of the array.

The second 3D image shows a quicksort algorithm. In this visualization the small blue boxes to the right represent the elements being sorted, the position of each blue box on the y axis indicates the element's relative value, and its position on the z axes (depth) indicates the elements position in the array. The multicoloured "exchange" planes to the left of the blue boxes illustrate the algorithm's
Figure 4.38: Two screen shots showing 2D and 3D Polka visualizations. The 2D visualization on the left shows the execution of a parallel quicksort algorithm, this visualization contains a control panel (top), a blocks view (left, height = value, horizontal position = position in array), and a chart view (right, vertical position = execution time, horizontal lines = swapping elements). The 3D visualization shown on the right shows the execution of a quicksort algorithm in a single 3D view (y axis = element value, x axis = execution time, z axis depth = position in array).

Further information on Parade and Polka can be found in [Stasko, 1995], [Stasko and Kraemer, 1992], [Stasko, 1994], or on the world wide web, see http://www.cc.gatech.edu/gvu/softviz/SoftViz.html
http://www.cc.gatech.edu/gvu/softviz/parviz/polkaanim.html

4.2.3 Information Visualization

Shneiderman, Osada and Ahlberg - Dynamic Queries

“Dynamic Query Interfaces” seek to apply the principles of direct manipulation to database query methods [Shneiderman, 1994]. Shneiderman identified four defining features of dynamic queries:

1. The visual presentation of a query’s components and results.

2. Rapid, incremental and reversible control over a query.

3. Selection by pointing rather than typing.
CHAPTER 4. REVIEW OF RELATED WORK

Figure 4.39: The FILMFINDER system which uses alphasliders to identify; film titles, leading actors, leading actresses, directors, and the film length. The x axis is used to indicate the year of release and the y axis indicates its popularity through cinema ticket sales.

4. Immediate and continuous feedback.

An example application which uses the dynamic query approach is the "FILMFINDER" system [Ahlberg and Shneiderman, 1994]. In FILMFINDER a database of film details are accessed through the use of alphasliders and buttons, with the resulting information being displayed in a 2D scatterplot. An example screen shot of the FILMFINDER system being used to find a selection of films staring Sean Connery is given in Figure 4.39. This and other FILMFINDER views are available on the world wide web, see http://www.cs.chalmers.se/SSXII/ivee-dumps/filmdirner.html

An "AlphaSlider" is an example of a dynamic query interface [Osada et al., 1993]. Continuous feedback keeps the user informed of their current position within the data set. A rectangular button is used in a range-defining alphaslider to identify a range of interest. Dragging the left and right hand edges of the rectangular button defines the start and end of the data range, and the rectangle itself can also be dragged to pan across the data set.

A selection of some of the work done using dynamic queries can be found in Christopher Ahlberg's world wide web site on information visualization and exploration, see
http://www.cs.chalmers.se/SSSKII/livee.html

More recently a project exploring the use of dynamic queries for SV has started at Washington State University; further information can be found at the exploratory visualization world wide web site, see

http://swarm.cs.wustl.edu/~roman/QueryVis.html

4.3 Summary of the Contributions Made

In this chapter the existing visualization support tools and techniques suitable for displaying the key characteristics of GAs have been introduced, and their suitability for supporting the user's understanding of the GA's search behaviour has been discussed. The final summary draws together the contributions made and remaining work to be done.

The conclusions of the user study highlighted a need to support the user's understanding of the GA's search behaviour. Of the key characteristics discussed, visualizing the GA's sampling of the search space is most effective for illustrating the GA's search behaviour. Although measures of the populations' diversity or problem complexity may be useful to indicate the GA's search behaviour, actually seeing the GA's search behaviour gives the user a more direct insight. The only problem with this approach is representing the high dimensional search space on a two dimensional screen.

Other key characteristics of significance for this project are the navigation of the GA's execution and visualizing the quality of the GA's solutions. The provision of bi-directional navigation support for viewing the GA's execution generation by generation and the potential use of dynamic queries for exploring sections of the search space, are two new approaches for GA navigation which have proven to be extremely useful within the respective fields of SV and information visualization. Visualizing the quality of the GA's solutions using a fitness versus time graph is the most common form of GA visualization simply because it shows the GA user something that they need to know. The provision of a fitness versus time graph is an essential view that can be augmented either with a vertical line to highlight the current generation, or with a rectangle to highlight the range of generations and fitness ratings being displayed in other views.

Editing the GA's parameters and operators may be useful for the GA user yet it is not directly a
part of the GA's visualization, however, if visualization support for understanding the GA's search
behaviour is not available then it will be difficult for the user to judge the effects of any changes made
except those which directly influence the quality of the final result. Editing the chromosomes in the
GA's population is not a common step involved in using a GA, but it may be a useful way of intro-
ducing problem specific knowledge or exploring the GA's behaviour. Although human-intervention of
this nature could be considered intrusive or even damaging to the GA's operation, such arguments are
outside the scope of this project, where the primary concern is for supporting the GA user. Support-
ing the editing of the GA's chromosomes may be achieved by providing an interactive search space
visualization. This could be used to explore unconsidered sections of the search space independently
of the GA's population, seeding the population with specific chromosomes, or (if the user wishes) for
moving chromosomes in the population to new positions in the search space.

Visualizing the chromosomes' (genotypes or phenotypes) involves the use of detailed (problem-
independent or problem-dependent) views of the solutions being considered by the GA. Viewing all the
chromosomes in a population produces a lot of information and unless the user is specifically interested
in examining the population's chromosomes (as they are with V1s in the Virtual Virus project), such
visualizations should be used selectively so that the GA user can identify the individuals that they
are interested in. The last key characteristic examined was visualizing the GA's operators, which
is an effective educational visualization but is not an informative visualization of the GA's overall
search behaviour.

In conclusion, visualizing the GA's sampling of the search space, navigating the GA's execution
and coverage of the search space, and displaying the quality of the solutions found by the GA, are
the three most important forms of visualization support for the user wanting to understand the GA's
search behaviour, the precision of which is discussed in the next chapter.