Chapter 5

The Popup Workshop System

In the previous chapter, the basic types of elements, the building blocks of pop-ups, were introduced since any attempt to provide design tools for paper engineering must be based on these elements. In addition, the methods used by other researchers in this area were examined. It has been shown that these methods are not aimed at children, nor are they proposed as a learning tool for this craft. In addition, it was seen that the only available software is designed only for the very limited subset of origamic architecture and not for pop-ups in general and that more general pop-up software is not available. Finally, some broad proposals were made as to how software for children learning the craft of pop-up making should function.

This chapter proceeds from that discussion and details the design and construction of the Popup Workshop system. Section 5.1 builds upon the general requirements for a pop-up design system for children developed in Section 4.3 to present the overall criteria for the design of Popup Workshop. An outline of the version history and the reasons behind the selection of Java as the programming language are reviewed in Section 5.2. Section 5.3 discusses the software from the viewpoint of the user: the windows, tools, and menu items. Section 5.4 is concerned with the internals of the software including the data structures, the geometric constraints that must be maintained for each element, and the algorithm used in opening and closing a page.

5.1 High-Level Design Considerations

Section 4.3 introduced several general principles that are embodied in Popup Workshop. First is the desire is to emulate the actions of a paper engineer as closely as possible. This means that the focus of the process should be on selecting elements to produce the desired design and motion, and then drawing each element's shape on the pattern rather than on a 3-dimensional representation, as a paper engineer cuts or draws her design on a flat piece of paper. But it also suggests that a 3-dimensional representation be made available that can be easily manipulated to see what the motion of the pop-up is like before the design is printed, glued and folded. Second, by enforcing geometric constraints on the elements, it is possible to insure that any design can always be folded flat. Third, the user should be able to change and delete elements easily in order to allow experimentation with many designs and their variations. Fourth, the operation of the software should be as simple as possible, offering ample help and with minimal or no explicit error conditions presented to frustrate the user. Fifth, the most commonly used elements should be available in the software while allowing them to be decorated, modified, or augmented when the pop-up is constructed. This helps the user build the knowledge and skill necessary to grow beyond the capabilities of the software. Sixth, easy to remember names for the elements should be used consistently in all parts of the software. And finally, standard file formats should be used in order to allow users to share designs or to bring them into other programs for decoration. In this section, some specific design decisions embodied in Popup Workshop are described that extend or complement these general principles.

The decision to include limited numbers of the most useful elements in Popup Workshop needs to be further explored. While there is the desire to add more elements, thus expanding the capabilities of the software, it also helps learning to have some things that cannot be done by the program, forcing the learner to do them. In the design of Popup Workshop, many elements and their variations were considered. However, it was decided to include only a few of the most widely used elements. This allows children the opportunity to modify those elements into new forms

for themselves while minimizing confusion from having to chose between similar elements when designing a pop-up. 90° elements are useful for origamic architecture and mathematics education and are more apt to be already known to children from any previous pop-up making experiences. 180° elements are used extensively in traditional pop-up books with which children are familiar. Popup Workshop thus contains both of these types of elements in order to allow children access to those elements that may have been encountered previously. In both of these categories angled and parallel elements are represented giving users the opportunity to create more varied and interesting pop-ups. To this end, five elements were chosen for Popup Workshop: the beak (angled 90°), the step (parallel 90°), the angled step (angled 90°), the tent (parallel 180°) and the v-fold (angled 180°). It is interesting to note that in Table 4.1, tents, v-folds, and beaks account for approximately half of all of the forms used in the books examined.

Some important elements were not included. Elements that utilized bent paper shapes were considered beyond the scope of this work, as they introduce complications in modeling. A major assumption in modeling the 3D shape was that distances of points on a plane of an element remain the same during the opening of a pop-up, something that does not occur with bent paper. Attached planes (see Figure 4.9) come in many forms and can easily be added after the pop-up is assembled, as they are most often flat pieces of paper glued to the side of another element. A tent can be used for the supporting structure to hold an attached plane away from the side of an element (see Figure 4.9). There are also additional elements that are easily constructed by modifying those that are provided. For example, moving arms are indirectly available by adding an attached plane to a beak. As mentioned earlier in this chapter, the v-fold can be converted to an angled tent by the method shown in Figure 4.15. Indeed, there are many possible variations on v-folds, some of which could have been included as separate forms, but all of which can be constructed using the provided simple v-fold as a starting point.

With respect to the design functions of the software, there were capabilities that were not included for the same reasons. For instance, only two decorating options were included: the painting of planes in a single color and the drawing of colored lines. This is in no way limiting as

the designer is free to further embellish her pop-ups in a variety of ways. Pop-ups can be imported into other programs to be decorated with textures or photos, straight lines can be cut as curves, and surfaces can be decorated with various art materials or have additional pieces added to them.

Although the software tool aids the design process, the final jobs of cutting, folding and gluing were left for the user, even though it is possible to use a laser cutter or a knife machine to cut out pop-ups.[1] This, however, is not desirable as children need to learn how pop-ups are put together and gain experience with tools and materials.

One of the general principles mentioned previously was to minimize user encounters with error conditions. Trade-offs must be considered in decisions involving error reporting. One of the ways in which beginners learn any skill is by making mistakes, discovering their mistakes (or having them pointed out), and correcting them. On the other hand, if errors are constantly reported, the beginner can become frustrated with the process. There are two questions that arise. First, should the user be able to produce a non-functional pop-up with the software? Second, if such problems are allowed to occur, should the user be alerted to their presence? It has been noted that there are two ways in which the design of elements can produce a pop-up that does not open and close properly: not adhering to geometric constraints and the collision of multiple elements. Using a computer for design allows the geometric constraints to be checked and enforced. Therefore, it seemed beneficial to silently enforce constraints in software in order to eliminate this class of errors. This decision removes the opportunities for the user to be forced to either learn the constraints or to produce non-functional pop-ups, but it makes the software much more usable. Likewise, collisions could be identified, but the presence of a movable 3D representation of the pop-up allows the user to observe when a collision occurs in most cases. Thus, no error reporting was included in the software.

Concise and thorough documentation for the software can assist the user who does not have an experienced paper engineer on hand. It should include information about how to construct the final pop-up from the design produced, pointers to other sources of pop-up construc-

[1] A laser cutter was used by the author on one origamic architecture design as a test, and worked quite well.

tion advice, and photos of designs made by other user of the program. In addition to advice and sources for students of the craft, the documentation for Popup Workshop 2.0 (see Appendix C) includes curricular references for teachers as well.

In spite of the stated purpose that this software to be used by children, few design decisions were driven specifically by this purpose. There were two reasons for this. First, an overall desire already existed to make the software simple to use. Simplicity of use is not an issue only for children; simple software works for adults as well. Second, there was some uncertainty about the ages of the children who might use the software. Reports of pop-up making in the classroom (see Section 3.3.2) included children as young as six for Johnson [58] and as old as high-school aged for Simmt [108]. However, there was some doubt about the range of these ages for general pop-up making as very young children might find the craft too difficult and older children might think that it was "for babies".

There were however a few decisions based on children using the program. It has already been mentioned that error conditions were avoided or left to the user to detect in order to minimize frustration. It was also decided that the ability to read should not be a prerequisite to using the program. For this reason, buttons are labeled with icons while the tool-tips, which offer supplemental information, use the written word. Consistency of operation was a major design consideration. For example, all elements are added in the same way, and the change, delete and replicate functions use the same sort of operation, with small squares appearing on points for the user to choose. Finally, although x- and y-coordinates for the mouse are given, a grid may also be used for positioning corners of the elements. Thus, understanding Cartesian coordinates is not necessary.

5.2 Design History of Popup Workshop

Java was chosen as the programming language for Popup Workshop primarily to provide portability. Design was done on the Macintosh, but it was desired that a user could use any operating system on any personal computer. In addition, Java is freely available and has a large

user community. This last consideration is important for future development as it is hoped that the software can be released to the open source community. Finally, Java has (in Swing) a graphical interface environment in which the use of the mouse for drawing could be supported, and the Java 3D libraries simplified the process of showing animated pop-ups. Java was used from the beginning of the design for Popup Workshop.

Version 1.0 of Popup Workshop was made for the Macintosh only and included just the three 90° elements: the step, the beak and the angled step. Since only 90° elements were included in the program, there were no extra pages for attached pieces. The graphics in the Viewer Window were somewhat primitive as perspective was calculated for a static object. Although the pop-up could be opened and closed, no rotation was available. All of the capabilities of editing pop-ups were provided (change, delete and replication of elements) and all decoration options were present, and therefore the interface looked quite similar to later versions. The File menu items were not present, therefore no capabilities to save or load files or to export graphics were available. Printing had to be done by capturing the screen. The points of the elements used for opening and closing the pop-up were computed using a constraint system, but the algorithm used random hill-climbing. This provided acceptable results, although it was slow when many elements were on the pop-up. Although Version 1.0 was crude, it was capable of allowing designs to be made, and proved the viability of constraint programming for the calculation of opening and closing the pop-up. Version 1.0 also provided a proof of concept for enforcing the geometric constraints of elements. Several test cards were made with the program, and it was decided that future versions would include only minor changes to the user interface.

In Version 1.1, support was added for Windows, a faster algorithm was implemented for for opening and closing the pop-up animation, and an x- and y-coordinate display was added to the Editor Window. Probably the most important addition was the **File** menu that provided the ability to print pop-ups, and to save them in forms that could be opened both in the software, and as graphical images. These additions required the selection of appropriate file formats.

Section 4.3.3 discussed the desirability of saving designs in a compact, text-based format

that allows the user to revisit previous designs to change or improve them or to share designs with other users of the program. Extensible Markup Language (XML) was chosen for Popup Workshop as it is a flexible, readable format that can be adopted to the task of saving pop-up designs. In addition, Java provides classes for parsing XML. That it is human-readable is an advantage as well for the developer, as it allows examination of the pop-up designs produced and provides some debugging ability. More detail on the format of Popup Workshop save files can be found in Section 5.4.2.

It is also useful to be able to share designs with those who do not have the software or to print designs without using the software. In addition, a standard graphical format can be imported into software tools for additional decoration, size changes, or other alterations. Many graphical formats would be appropriate, but the Joint Photographic Experts Group (JPEG) format is a standard, nonproprietary format that can be displayed and manipulated by many graphical tools.

Aside from these improvements, the program was largely unchanged in Version 1.1, as the interface had been found useful, and the 90° elements previously included were kept. Some small amount of user testing (further described in Chapter 6) was done with a few 5th grade students, some middle school students in a summer program, and an undergraduate intern. During that testing, it was realized that better animation that could be viewed from multiple angles would help users identify and avoid collisions between elements. Also, the lack of 180° elements, while not crippling, did not allow children to produce pop-ups that looked like those they were used to seeing in professional pop-up books. It was also realized at this time that messages to suggest what action might be taken after a button was selected would help the user, as the students (who were old enough to read) had difficulty with the click and drag method of drawing for the first few elements they made.

Building on this information, Version 2.0 added two 180° elements and took advantage of the Java 3D libraries to completely rewrite the viewer. This allowed for rotation of the pop-up in the viewer and the addition of more realistic lighting made it easier to see what elements were

doing when the pop-up was in motion. Since the elements could now be viewed from any angle, it became necessary to cut out those areas in the planes that the 90° elements produce. Help was added, and the terms used for elements were standardized to their current form. Version 2.0 was used for the testing described in Chapter 6 and is detailed in the following sections.[2]

5.3 User Interface

There were several design decisions that affected the interface and should be enumerated here. First, the design of the pop-up is created on the 2-dimensional representation (the pattern), and the 3-dimensional representation reflects those changes at the same time. This allows the user to learn the notation for 2-dimensional patterns and also to see the design as it will exist when it is constructed. The mouse is used to add and manipulate elements in the pop-ups. Existing design systems use either the keyboard (Pop-Up Card Designer [113]) or the mouse (Glassner [36, 37]). The keyboard option seems only usable for the simple case where a single element (step) is being created. Since the desire here was to allow the user to create several types of elements and easily manipulate them, the mouse allows more options and control.

Some of the more important interface decisions concerned the methods for adding elements and for editing elements already present. All actions are chosen by selecting buttons that are grouped into three tool palettes. In the first tool palette, the button corresponding to the desired element is selected and the element is then drawn using the mouse. To do this, the cursor is placed near a fold or seam (it does not have to be directly on the fold or seam), the mouse button is held down and the corner of the element is dragged into position. Only one side of the element needs to be drawn as the element produced is symmetric. The simple shapes drawn are a rectangle for parallel elements and a triangle for angled elements, thus reinforcing the differences between these forms. Elements can be altered later to be made asymmetric if desired. The second palette of buttons allows for changes to be made to the elements. These buttons allow the user

[2] Both versions 1.1 and 2.0, and the documentation for each, are available for Macintosh and Windows users on the Popup Workshop web page [48].

to change the position of an element's corner points, as well as delete or replicate an element. Once again, all the operations in this palette operate in the same way. Small squares appear on all corners of the elements and the user can pull them (for change) or click them (for replicate and delete) to effect the desired action. The third palette contains tools for decoration. These include buttons to change the fill or drawing color, fill planes with color, draw lines, and show or hide the grid lines.

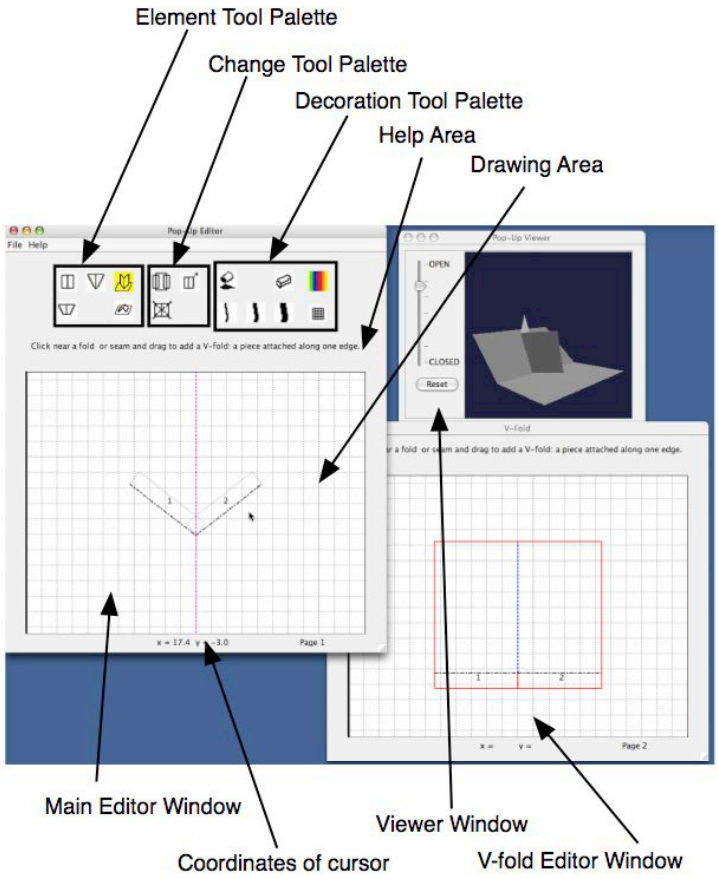


Figure 5.1: Popup Workshop Windows and Palettes: Popup Workshop with the user adding a v-fold to the base page. The base page is in one Editor Window, with the v-fold that will be attached to the base page in another. The Viewer Window displays the 3-dimensional representation of the pop-up.

The interface has been kept simple without being specifically oriented to younger children.

Simplicity allows reduced cognitive load and quick learning of the interface. Actions and results have been made as consistent as possible. For instance, all elements are drawn by placing the mouse near a fold or seam, then clicking and dragging. Change, delete, and replicate all manipulate the same small boxes to produce their result. Menus are kept standard (with respect to other software that the users may have encountered) whenever possible. Help and documentation use a simple name for each element, and the names are used consistently.

Figure 5.1 shows an example of Popup Workshop being used to design a pop-up with a single v-fold element. The following sections give an overview of the operation of the software from the user's perspective.

5.3.1 Editor Window

The Editor Window has a drawing area where the pattern for the pop-up is drawn as a 2-dimensional form. This area has a grid for positioning elements that can be toggled on and off, and displays the x- and y-coordinates of the cursor when it is within the drawing area. This coordinate system places the origin at the center of the 8.5 inch by 11 inch drawing area with each grid square representing 0.5 inch. Because of the screen resolution, the actual size of the drawing area on the screen can vary, and will be scaled at print time to fit within the print area of the printer used.

There are no error messages produced by the user's actions in the Editor Window as each element is limited by geometric constraints that are constantly enforced. For instance, trying to make an element so big that it does not fit on its base planes will result in the element being made only as big as will fit. Trying to add an element that is too small, for instance one with a zero-length width, will result in no element being added. The click and drag operation for adding elements only works when the starting point is near an acceptable fold or seam. This means, for example, that a user will not be able to draw a 90° element on a seam, since this would require a cut be made across the glued line.

There is always at least one Editor Window. The main Editor Window appears when the

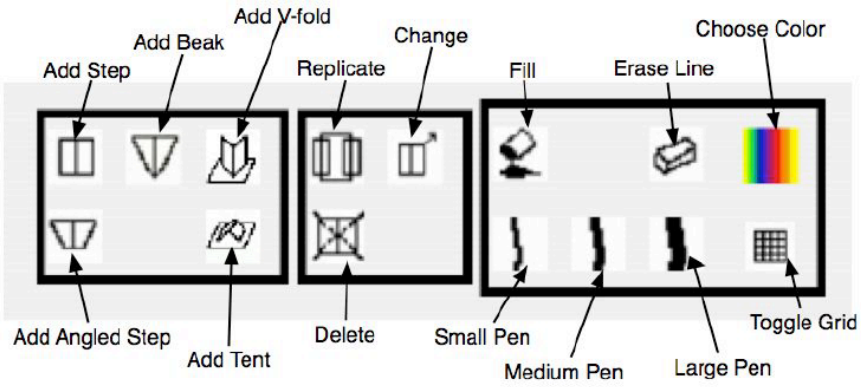


Figure 5.2: Popup Workshop Palette Buttons: The buttons available in the tool palettes of Popup Workshop. The left palette contains buttons for adding elements, the center palette buttons allow changes to the design, and the right palette is for decoration operations.

program starts, and persists until the program terminates. In addition, there will be an additional Editor Window created to display each extra piece for 180° elements if any are added. These extra windows act just like the main Editor Window in that elements and decoration can be added, changed, and deleted. For instance, in Figure 5.1 an element could be added to the center fold of the v-fold in the V-fold Editor Window, to the gutter in the main Editor Window, or to the seams of the v-fold in either window. Each Editor Window has a page number and any extra pieces are given numbered gluing tabs that match the numbers on the corresponding gluing bases to which they attach.

Within the drawing area, cuts are represented by red solid lines, valley folds by magenta dashed lines, and mountain folds by blue dashed lines. This use of dashed lines for folds and solid lines for cuts is a standard notation for paper engineering patterns, although the color choices to indicate the direction of folds may be unique to this software and colors were chosen to be obvious and easily visible when printed. As there is no standard notation for seams, they are indicated by dashed black lines.

There are three tool palettes located at the top of the main Editor Window that contain buttons labeled with icons to represent their actions. When the user moves the cursor to a button

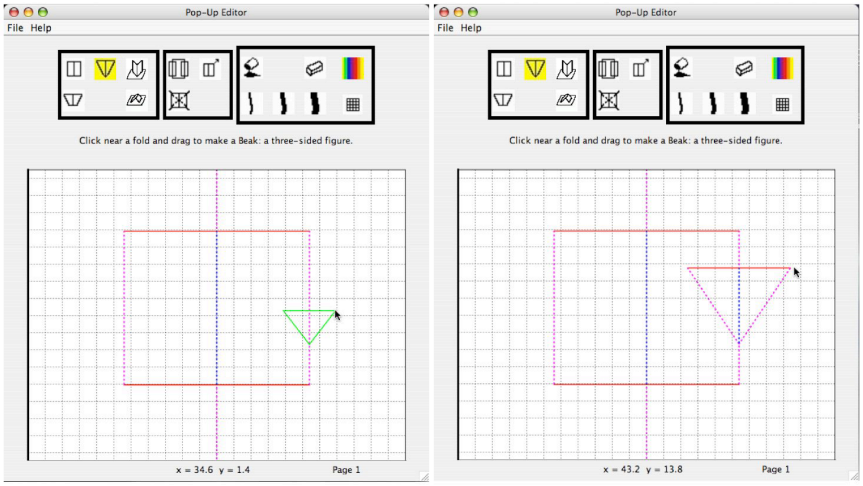


Figure 5.3: Adding an Element: Adding a beak element by clicking near a fold, dragging the outline to the desired size (left) and releasing (right).

a tool tip appears to explain what the button does. When a button is selected, a message appears in the help area that tells the user what action the program expects. Figure 5.2 shows the buttons available in the tool palettes. Extra windows for additional pieces do not have separate tool palettes but use those in the main Editor Window.

The Element Tool Palette (left in Figure 5.2) contains buttons that allow the user to add an element. The available elements are the step, beak and angled step (90°) and the v-fold and tent (180°). Figure 5.3 shows the process for adding a beak element.[3] When the button is selected it changes color. The user then positions the cursor near a fold (or a seam if the element can be placed on a seam) and clicks and drags to draw the outline of the new element. As detailed in Section 5.4.3, the new element is added with its geometric constraints and starting heuristics (symmetry, for instance) satisfied.

The Change Tool Palette (center in Figure 5.2) contains buttons for moving points of an element, deleting elements, and replicating elements. For each tool, small boxes are drawn at

[3] Examples of the action of various buttons is shown for 90° elements as they are simpler. The only difference for 180° elements is that a new window is added or removed as appropriate. Changes and additions may be made in the window containing the extra piece in the same manner as they are made on the main page.

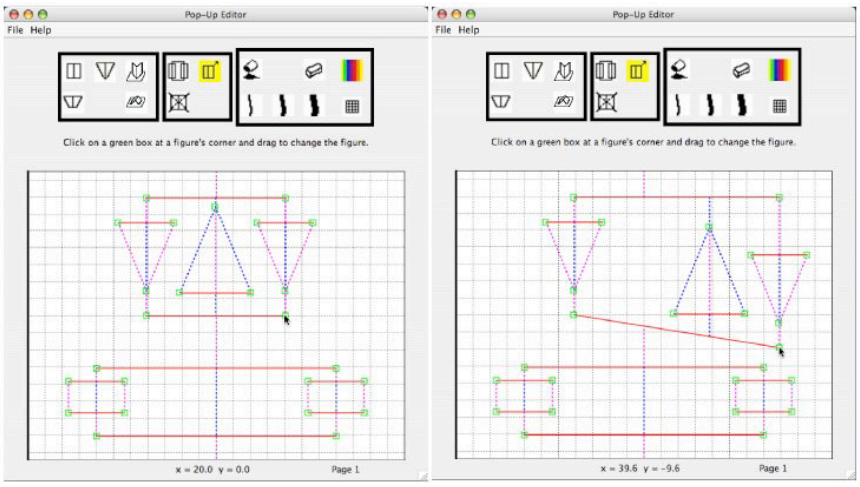


Figure 5.4: Changing an Element: Dragging the green handle on a corner of the element moves the point. Constraints are continuously satisfied, and any elements on top of the changed elements are also changed to remain on their parent folds.

corners of the elements in the drawing area that serve as handles to modify or select an element.

When the Change button is selected, green handles appear on the corners of all elements in the current design. The user can click on one of these handles and drag it to the location desired. The constraints for the element are automatically satisfied as the element changes. For instance, in Figure 5.4 the location of the point on the same side of the parent fold as the point being moved also changes in order to keep the folds parallel. The center fold also moves if needed to keep the element foldable. In addition, all elements on top of the changing element are moved so that they remain on their original parent folds.

When the Replicate button is chosen, blue handles are placed at the elements' corners. By clicking on any of these handles, the element is replicated on all folds of the element under it that are of the same type as its parent fold (valley fold, mountain fold, or seam). Figure 5.5 shows the replication action being applied twice, first for the larger box, and then for the smaller box that sits on it. In this example, the chosen element is replicated on all of the valley folds of the element below as well as being replicated on all additional copies of the element below it. If there are no matching folds (the element being replicated is on a center fold, for instance) or the element has

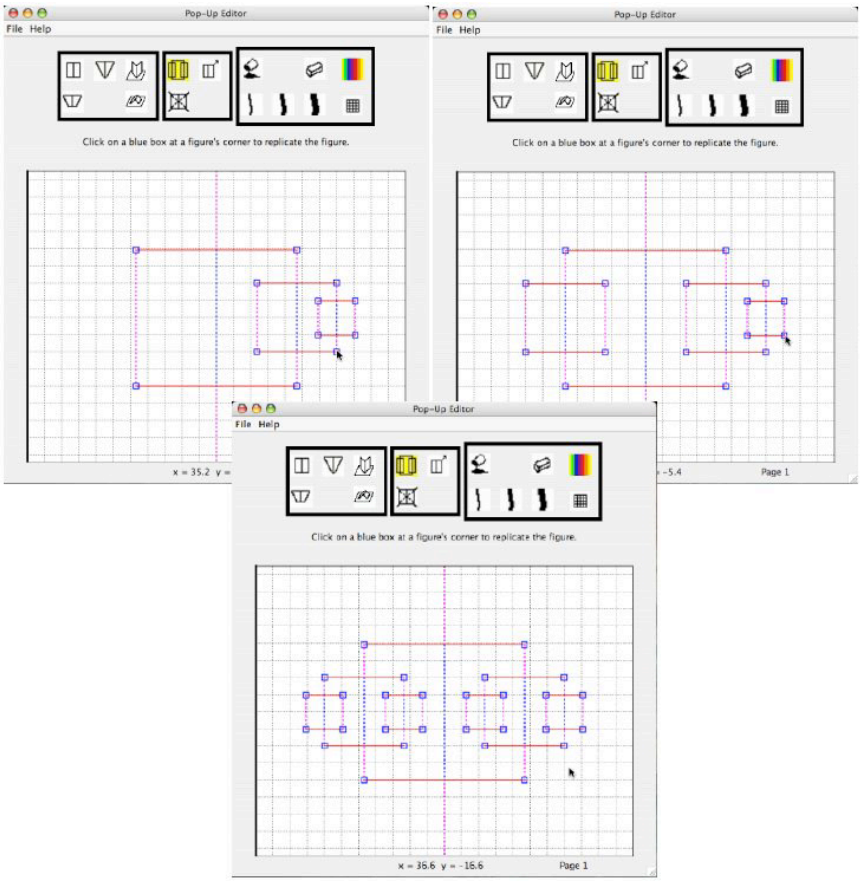


Figure 5.5: Replicating elements: Clicking on the blue handle replicates the element onto each matching fold of the element beneath it, in this case each valley fold.

already been replicated, clicking on the blue handle does nothing. The presence of additional elements on the matching fold does not inhibit the replication.

When the Delete button is chosen, red handles are placed on the elements' corners. By clicking on one of these handles, the element chosen is deleted. Figure 5.6 shows the deletion of the lower large step. If an element is deleted, all elements on its folds and seams must also be deleted, as well as the elements on their folds and seams, and so on.

The Decoration Tool Palette (right in Figure 5.2) contains buttons for filling areas with color, drawing lines, erasing lines, selecting the color for fill or lines, as well as displaying or hiding the grid.

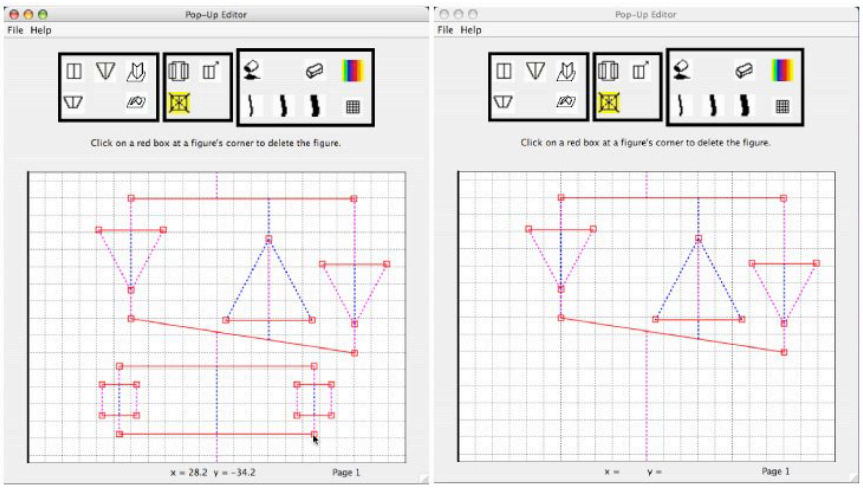


Figure 5.6: Deleting elements: Clicking on the red handle deletes the element and any elements on top of it.

When the Color Selector button is chosen, the color picker is displayed. This palette is standard for Java applications and provides several ways in which to indicate a color. The number of colors is dependent on the current display settings. When a new color is selected it becomes the border color for all tool palettes, and will be used for both drawing lines and filling areas.

The Grid Toggle button turns the grid in the drawing area off or on. Turning off the grid is desirable when printing the pattern.

There are three buttons to select drawing tools that produce lines of different widths using the currently selected color that can be drawn with the mouse. The Eraser button adds red handles on the ends of all lines and a handle is clicked to remove the chosen line.

The final tool is the Fill button. Any plane of an element or the base page can be filled with the currently selected color by choosing this button and clicking on the plane. Fill colors can be erased by filling the area with white.

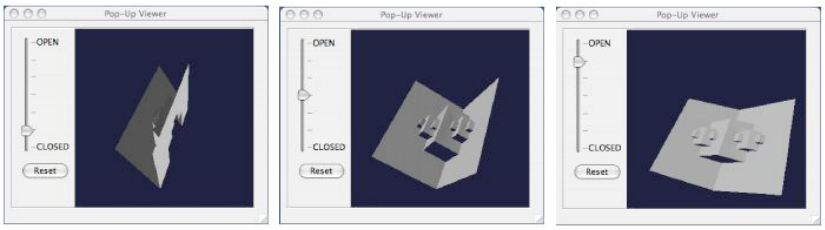


Figure 5.7: Viewer Window: The the slider control can display the pop-up in any position from completely open to completely closed as shown in these three examples of the motion of one pop-up. The Viewer window provides real-time information about the motions of the elements. The pop-up can also be rotated to any orientation.

5.3.2 Viewer Window

The Viewer Window displays a 3D representation of the current pop-up that can be manipulated by the user. Clicking on the 3D image and dragging the mouse will rotate the pop-up in any axis to allow the user to view it from any angle. The pop-up can also be opened and closed by using the slider control at the left side of the window. Figure 5.7 shows a pop-up being opened and closed in this manner. Any manipulations can be undone and the image returned to its default starting position (open 90° and facing directly toward the user) by clicking the Reset button below the slider.

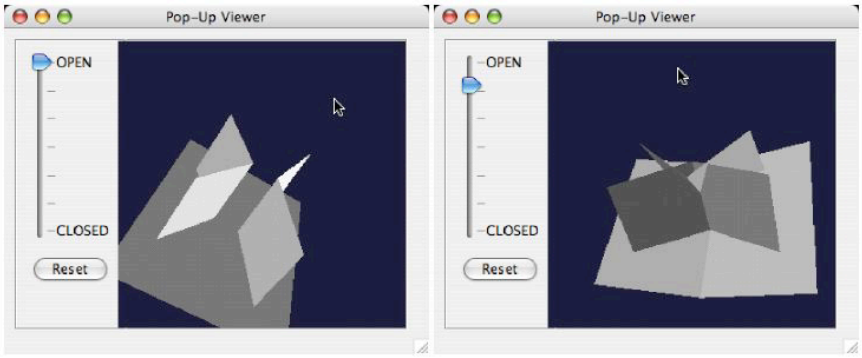


Figure 5.8: A Collision Between Elements: This is an example of a case where collision between elements occurs. When fully opened, the v-folds do not collide. Use of the open and rotate functions make the problem obvious.

In Popup Workshop, a pop-up that does not close correctly can be produced if there are collisions between elements. These will usually be obvious in the Viewer Window, as shown in Figure 5.8. Collisions are allowed for two reasons. First, to help the user learn to identify and judge such conditions. Second, it is possible that the colliding elements can be altered as they are being assembled, and that they may produce exactly what the user desires.

5.3.3 Menu Operations

To simplify the operation of Popup Workshop, all menu items are grouped into only two menus: **Help** and **File**.

Currently, the **Help** menu has only one menu item, **About**. The About Box displays version and developer contact information. The menu exists primarily as a placeholder for future help functionality such as providing a searchable version of the documentation.

The **File** menu contains menu items that are common to the File menus in other programs. In general, menu items are only available if they make sense in the current state of the design process, and the user is given the opportunity to save the current pop-up if a choosing a menu item would cause the current work to be lost.

New, the first menu item, creates a new pop-up from scratch. In any situation in which the current pop-up has been changed since the last save, the user is asked if she wishes to save the current pop-up before replacing it. **New** is not available if the pop-up has just been created and is blank.

Open opens a saved pop-up. This allows a user who has previously saved a pop-up to continue work on the pop-up later, or to share it with other users. **Open** is always available. If changes have been made since the last **Save**, the user is asked if she wishes to **Save** when **Open** is chosen.

Save, and **Save As** are the next two menu items, and create files for the current pop-up that may be accessed with **Open**. **Save** is only available if changes have been made in the pop-up since the last **Save** or **Save As**. **Save As** is always available. For instance, if the current pop-up was

not previously saved **Save** is not available, but **Save As** is.

Export Image saves the Editor Window contents as an image. There may be more than one image saved, as each additional piece is stored in a separate image. A user can export a pattern created with Popup Workshop and later print the pattern from a computer that does not have Popup Workshop installed by opening the image in a graphics program. **Export Image** is always available.

Revert to Last Saved allows the user to delete all changes since the last **Save**. This menu item exists only if a saved version of the current pop-up exists. The user is warned via a dialog box to indicate that the current pop-up will be changed to the older version so that she can confirm the operation.

Print prints each page of the pop-up design onto a a separate piece of paper. This option is always available.

Exit terminates the program gracefully and is always available. The user is given a dialog box allowing a **Save** if any changes have been made since the last save.

5.4 Program Design and the Nature of Pop-ups

One important consideration in the design of Popup Workshop was to allow for future expansion and, in particular, that adding new elements should be as easy as possible. This requirement was applied to the selection of data structures, to the organization of the elements' class structure and methods, and to the selection of a method for animating the pop-ups.

In addition, there were no deliberate attempts made to optimize the code for speed or to refactor the code. Thus, there are blocks of repeated code, for instance between an element and its corresponding inverted form. However, the desire was to make it good enough, knowing that these early versions of the program do not have to be extremely small or fast. In practice, most pop-ups that the children made had only a few elements, with a very complicated one having at most 7 or 8 elements. In user tests, the most complicated pop-up made was Daisy's alien (Figure F.1) and it proved difficult for her to cut and fold. Even this pop-up with 26 elements did not

cause any speed problems for the user when opening and rotating it in the viewer, although the operation was visibly slower than on simpler examples.

Note that there are two uses of constraints in Popup Workshop, and one should be careful to distinguish between them. First, there are geometric constraints that apply to each element type. For instance, all the folds on a beak must meet in a single point. These geometric constraints must be considered when adding and changing an element. Second, for the opening and closing of the page Popup Workshop uses a constraint system to calculate the point locations for each element, instead of using a direct mathematical calculation.

The following sections discuss the most important details of the internal design of Popup Workshop. The first, Section 5.4.1 discusses the data structures and classes used, as well as the overall design. Section 5.4.2 provides details of file formats chosen for both graphical output and saved files. Section 5.4.3 discusses the geometric constraints that apply to each element type. Finally, Section 5.4.4 describes the algorithm used in the constraint system for opening and closing the page in the 3D viewer.

5.4.1 Classes and Data Structures

The selection of Java as the language in which Popup Workshop is written was made primarily on the criterion of portability. The desire was to produce software that could be easily run on many operating systems. However, the Java language proved a good choice in another way as well, as the structure and construction of pop-ups map neatly to Java classes. A pop-up is comprised of a set of elements, each of which has certain folds, cuts, seams, and geometric constraints. It is helpful to assume that any element "knows" how to open itself and change itself. The pop-up domain is also recursive, as the removal of any element forces the removal of all elements on top of it, and the change of any element means that elements on top may very likely have to be changed, as will any on the new changed elements. Java's support for this type of recursive construction was very helpful.

Popup Workshop consists of 43 Java classes. Some of these classes are used for the Swing

interface, for windows and palettes, or for specialized functions such as XML parsing. The remaining classes are concerned with the pop-up design itself and it is these classes that will be briefly outlined here, along with the most important of their members and methods.

Two classes exist to provide basic geometric components and methods. Point3D is a class representing a single point, and the class LineFeature connects two Point3Ds to represent a line segment. Point3D has x-, y-, and z- coordinates. For the 2D representation of the Editor Window the z-coordinate is always given the same value. Both Point3D and LineFeature contain methods that are used to obtain geometric values. For instance, the Point3D class contains methods to find whether two points are the same (to a given precision) and whether the point is on a given line. The LineFeature class contains methods that do similar operations for a line segment. Finding the midpoint, line angles and intersections, and determining if lines are parallel are a few of the methods provided for LineFeature.

The ValleyFold, Cut, MountainFold, and Seam classes inherit from LineFeature and are necessary to build an element. A great deal of the development effort in Popup Workshop was devoted to defining relationships between classes. For instance, a fold or seam needs to link to the two planes of the elements on either side of it. The LineFeature also has a method to return those planes.

Therefore, the Plane is another important class, and represents a plane on an element. The Plane contains a list of Point3Ds around the edge (in 2D coordinates) and the current coordinates of each of the points for the current angle of the page (3D coordinates) Planes are required to change shape, that is, cut themselves out, when a 90° element is place on them. A beak, for example, cuts a triangular section out of each plane on which it sits. The Plane class has a list of boolean values that indicate which points were in the original plane when it was built (not cut out.) These are necessary when the element on top is deleted or changed and the original outline of the plane must be restored.

Elements are represented by the Structure class, with AppliedStructure (for 180° elements) and SlitStructure (for 90° elements) inheriting from Structure. Each individual element class in-

herits from one of those two classes. There are 9 element classes: DoubleSlit (step), InvDoubleSlit (inverted step), and so on. Each of the element classes contains such data as the folds, cuts, and seams appropriate to the type, its Planes, and the Planes lying below it. All Structures contain methods to open, change, and delete themselves. Deleting and changing elements are recursive operations. The changed Structure finds all Structures on its folds or seams, and asks them to change as well and in turn, they do the same.

The main data structures are located in the class PopUp. This class contains linked lists of Structures (the current elements), ParentFolds (the fold on which each element sits in the same order as the elements), Planes (the planes making up the elements), and Pages (the base page and any pages added by the 180° elements). The list of Structures is kept in order by appending to the list when a new Structure is added, and removing Structures from the list when they are deleted. This allows a single pass to open all of the Structures to the current page angle. The ParentFold list is searched when replication is done. Since the Plane stores its 3D coordinates at the current opening angle, the Plane list is the source of current pop-up data for the Viewer. Java3D takes the list of points for each plane and builds the current image of the complete pop-up. The PopUp class also holds data applicable to the entire pop-up, such as the current page angle. The PopUp class initiates such actions as adding an element, changing the opening angle, and drawing the design to the screen.

5.4.2 File Formats

Section 5.3.3 described the menu items used to write files. The Export menu item exports the pop-up design in a graphical form so that it can be exchanged between users or imported into graphics programs such as PhotoShop. The Save menu item, on the other hand, saves the pop-up in a form that may be re-opened in Popup Workshop. Both methods allow users to share designs.

The Export menu item writes a file in the Joint Photographic Experts Group (JPEG) format. JPEG was chosen for its ubiquitous use as a file format in many graphics programs, common use on the Web, and the fact that a JPEG image can be easily written in Java. A bufferedImage class

can be painted exactly as the screen is painted, with 8-bit RGB color components packed into integer pixels. This buffer is then written to the file with class ImageIO.write, that allows "jpg" as a parameter. This produces an image of 800x650 pixels, with resolution of 72 ppi.

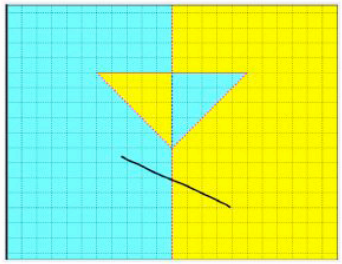
In order to save a pop-up for future work, Extensible Markup Language (XML) is a logical choice. Pop-ups are modular, consisting of elements with particular attributes (corner points, plane colors, parent folds) that lend themselves to a nested structure. In addition, XML files are relatively small text files, ideal for email and other modes of transfer. Some debugging was accomplished during development by examining save files, since they are human-readable. An additional reason for the use of XML is the existence of classes to parse XML in Java.

The organization of tags in a save file is shown in Figure 5.9. A pop-up is delineated by the *popup* tag pair. The version parameter indicates the version of Popup WorkShop writing the file. Within this pair, there are three areas. First, *mainpage* identifies the colors of the planes of the base page of the pop-up. There is no other information currently saved for the main page.

Second, a list of all the elements is delineated by the *structurelist* tags. Within this list, *structure* tags enclose each element. The type parameter is an integer indicating whether an element is a beak, inverted beak, step, etc.. An element consists of a list of *outerpoints* (the positions of the corners of the element in the order used in the element constructor), items designating the left and right plane colors, the left and right gluing tab colors, and the beginning and end points of the parent fold (so that the parent fold can be identified when the element is added), and the page on which the parent fold is located.

Third are the lines drawn as decoration. The list of lines is delineated by the *drawnlinelist* tag pair. The information for each line consists of the thickness of the line, the color, the page on which the line occurs, and a list of the points defining the line.

The elements are stored in this save file in the same order as in the linked list kept in the PopUp class. This allows the constructors for each element to be called in turn in the same order. Once the parent fold is located on the correct page, the outer points are the primary information used in constructing a new element. Since the elements are added to the save file in list order,



```
<?xml version='1.0' encoding='utf-8'?>
<popup version='2.0'>
  <mainpage leftcolor="-10027009" rightcolor="-256">
  </mainpage>
  <structurelist>
    <structure type="2">
      <outerpoints>
          <point>
            <x>-0.0</x> <y>5.2</y> <z>-105.0</z>
          </point>
          <point>
            <x>-25.0</x> <y>-20.0</y> <z>-105.0</z>
          </point>
          <point>
            <x>25.0</x> <y>-20.0</y> <z>-105.0</z>
          </point>
      </outerpoints>
      <leftplane>
        <color>-256</color>
      </leftplane>
      <rightplane>
        <color>-10027009</color>
      </rightplane>
      <lefttab>
        <color>-1</color>
      </lefttab>
      <righttab>
        <color>-1</color>
      </righttab>
      <parentfold>
          <point>
            <x>0.0</x> <y>-42.5</y> <z>-105.0</z>
          </point>
          <point>
            <x>0.0</x> <y>42.5</y> <z>-105.0</z>
          </point>
          <page>1</page>
      </parentfold>
    </structure>
  </structurelist>
  <drawnlinelist>
    <line thickness="13">
      <color>-16777216</color>
      <page>1</page>
          <point>
            <x>-16.6</x> <y>8.0</y> <z>-105.0</z>
          </point>
          ...
          <point>
            <x>19.2</x> <y>24.8</y> <z>-105.0</z>
          </point>
    </line>
  </drawnlinelist>
</popup>
```

Figure 5.9: Tags in a Popup Workshop Save File. The pop-up design in the Editor Window is shown on the left and consists of a beak and one line with fill colors on the planes. On the right is the resulting XML from a save. Some of the points for the line have been removed to simplify the example.

and processed in that order when the file is opened, the lower elements are in place before the upper elements are added.

5.4.3 Geometric Constraints of the Elements

Section 4.1.2 introduced the concept of geometric constraints required for proper opening and closing of elements. In Popup Workshop, these geometric constraints are satisfied for all elements at all times. Keeping the geometric constraints satisfied insures that the pattern always represents a foldable design and eliminates the need for a foldability check on the entire design. If an element is changed, those changes are constrained to keep the element foldable. For instance, in the beak element the center point must lie on the parent fold. This means that when the point is changed, the user must only be able to slide the point up and down that fold. Sometimes the change in a point will necessitate a change in other points. For instance, if a fold line must remain parallel to another fold line, and one endpoint of the fold line is changed, both end points must move in order to keep the fold lines parallel. Whenever a new element is added, the geometric constraints are satisfied. For instance, the center point of the beak is on the parent fold from the time the element is constructed.

In addition, there are certain starting conditions that are followed in Popup Workshop. These are not geometric constraints, but rather useful requirements that establish the initial form of the element. These starting conditions arise either because of the way the element is "drawn" on the pattern, or in order to allow for further changes. For instance, in sweeping out the beak element with the mouse, the cursor moves one of the side points. The other is moved automatically to produce an isosceles triangle. The newly produced beak, then, starts with equal angles at the sides; it is symmetric about the x-axis. Beaks are not constrained to be symmetric and the user can alter the element later to be asymmetric. Another example is in the production of the piece on the extra page of the v-fold. There are no constraints that affect the shape or size of this piece, other than the requirement that the seams must be the same length as those seams to which they are to be glued on the original page. The starting condition is to make the extra piece a centered rectangle with enough room on the extra page to make changes.

To simplify the geometry of the elements, Popup Workshop assumes that cut lines on the

elements are straight line segments, even though the user can vary their shapes at construction time.

Although there are 5 elements that the user sees, in actual fact there are 9 classes of elements in the code as four of the elements have inverted forms. The following sections indicate the geometric constraints, starting conditions, simplifications for Popup Workshop, and the conditions in which an inverted element will be used for each of the 5 elements that a user can produce.

5.4.3.1 The Step Element

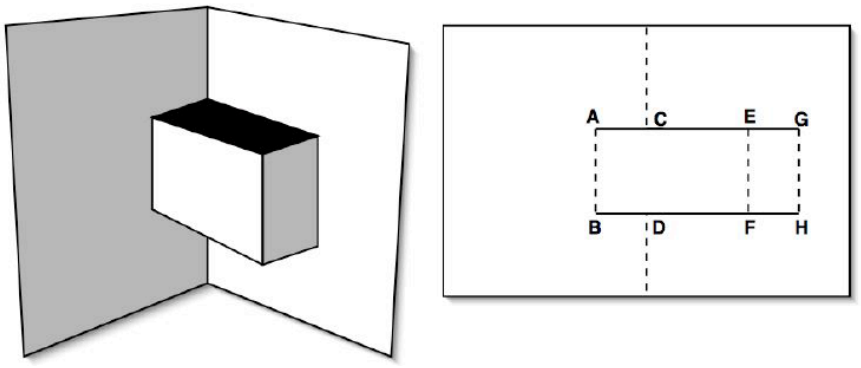


Figure 5.10: The Step Element in Popup Workshop, showing the points important for the constraints.

The step element (Figure 5.10) is a 90° parallel element. It is constructed with two cuts across the parent fold, two side valley folds, and a mountain center fold.

In order to fold properly, the basic constraints are:

- $\overline{AB}||\overline{CD}||\overline{EF}||\overline{GH}$
- The distance from $\overline{AB}$ to $\overline{CD}$ = the distance from $\overline{EF}$ to $\overline{GH}$
- Points A and B must lie on the left base plane. Points G and H must lie on the right base plane.

When an element of this type is added in Popup Workshop the starting conditions are:

- The distance from $\overline{AB}$ to $\overline{CD}$ = the distance from $\overline{CD}$ to $\overline{GH}$ and the beginning shape is a rectangle centered on the parent fold.
- Since the second constraint above holds, $\overline{CD}$ and $\overline{EF}$ coincide.

For the purposes of Popup Workshop, points A, C, E, and G lie on the same line segment, and points B, D, F, and H also lie on a single line segment. The inverted form of the step differs only in that the parent fold is a mountain, rather than a valley fold. This means that the two side folds are mountain folds and the center fold is a valley fold.

5.4.3.2 The Beak Element

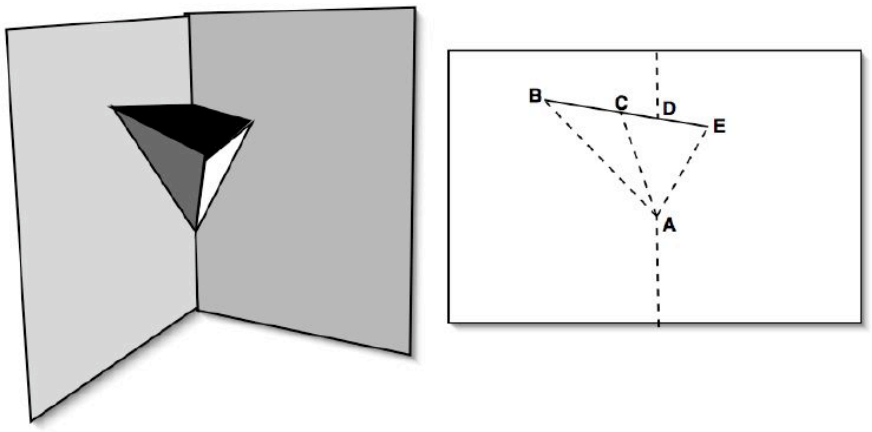


Figure 5.11: The Beak Element in Popup Workshop, showing the points important for the constraints.

The beak (Figure 5.11) is a 90° angled element. It is constructed with one cut across the parent fold, two side valley folds, and a mountain center fold.

In order to fold properly, the basic constraints are:

- Point A must remain on the parent fold, although it may move along it.

- $\angle BAC = \angle DAE$
- Point B must lie on the left base plane. Point E must lie on the right base plane.

When an element of this type is added in Popup Workshop the starting conditions are:

- Point C = Point D.
- The triangle BAE is an isosceles triangle, with $\angle ABE = \angle AEB$.

For the purposes of Popup Workshop, B, C, D, and E lie on the same line segment. Also, the triangle BAE can point up or point down, and this orientation can be changed. The inverted form of the beak differs only in that the parent fold is a mountain, rather than a valley fold. This means that the two side folds are mountain folds and the center fold is a valley fold.

5.4.3.3 The Angled Step Element

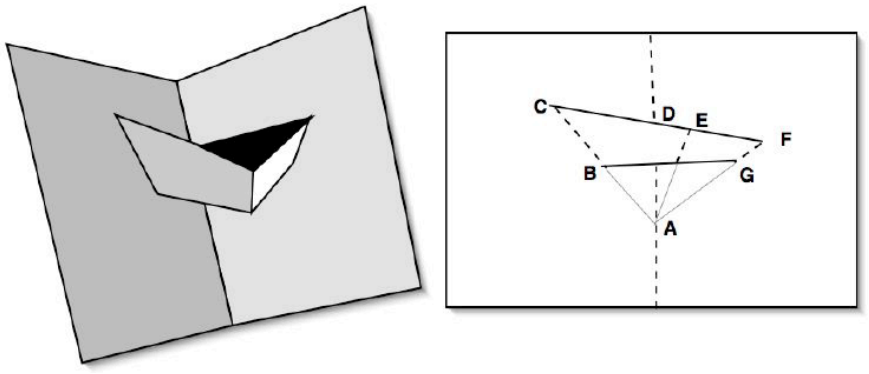


Figure 5.12: The Angled Step in Popup Workshop, showing the points important for the constraints.

The angled step (Figure 5.12) is a 90° angled element. It is constructed with two cuts across the parent fold, two side valley folds, and a mountain center fold. It is essentially a beak with the point removed and therefore many of the constraints are the same as the beak.

In order to fold properly, the basic constraints are:

- Point A must remain on the parent fold, although it may move along it.
- Point B must lie on $\overline{AC}$, and point G must lie on $\overline{AF}$, although they may move along the lines.
- $\angle CAD = \angle FAE$
- Points B and C must lie on the left base plane. Points F and G must lie on the right base plane.

When an element of this type is added in Popup Workshop the starting conditions are:

- Point E = Point D.
- The triangle BAE is an isosceles triangle, with $\angle ACF = \angle AFC$.
- Point B is the midpoint of $\overline{AC}$, and point G is the midpoint of $\overline{AF}$.

For the purposes of Popup Workshop, points C, D, E and F lie on the same line segment. Also, the triangle CAF can point up or point down, and this orientation can be changed. The inverted form of the angled step differs only in that the parent fold is a mountain, rather than a valley fold. This means that the two side folds are mountain folds and the center fold is a valley fold.

5.4.3.4 The V-fold Element

The v-fold (see Figure 5.13) is a 180° angled element. It is constructed with an extra piece glued to a v-shaped seam that is on either a mountain fold, a valley fold, or a seam. The extra piece has a mountain center fold.

In order to fold properly, the basic constraints are:

- Point B must remain on the parent fold, although it may move along it.
- $\overline{AB}$ is the same length as $\overline{GE}$, and $\overline{BC}$ is the same length as $\overline{EH}$, since the lines are the seams on which the piece is glued.

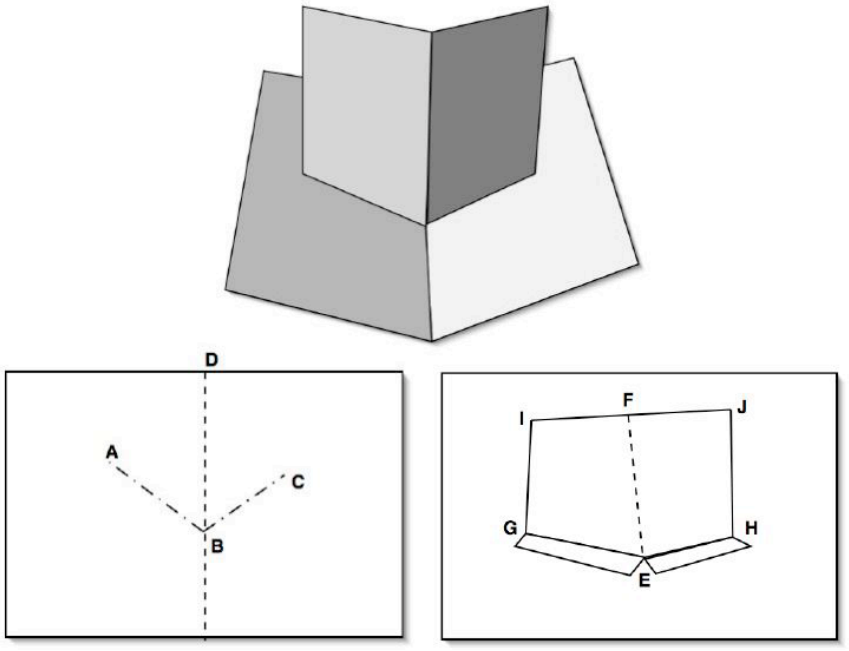


Figure 5.13: The v-fold in Popup Workshop, showing the points important for the constraints.

- $\angle ABC < 180°$. If the extra piece is glued straight across, the fold cannot move. In practice, if the angle is close but not equal to 180°, folding may be inhibited. Popup Workshop uses the rule that it must be less than 170°.

- $\angle ABD + \angle CBD < \angle GEF + \angle HEF$. Once again, if the two angles are too close, there may be folding problems. Popup Workshop keeps at least 10° difference in the two angle sums.

- $\angle HEF = \angle CBD + x$ and $\angle GEF = \angle ABD + x$ for some x.

- Point A must lie on the left base plane. Point C must lie on the right base plane.

When an element of this type is added in Popup Workshop the starting conditions are:

- $\angle GEH = 180°$.

- $\overline{AB}$ and $\overline{BC}$ are the same length, and $\angle ABD = \angle CBD$.

- The piece on the extra page is calculated to lie more or less in the middle of the page in order to give the user room for changes. $\overline{GE}$ and $\overline{GH}$ are known from $\overline{AB}$ and $\overline{BC}$. Points I and J are 1/4 of the page height down from the top margin, and Points G and H are 1/4 of the page height up from the bottom margin. The piece is centered horizontally.

For the purposes of Popup Workshop, points I, F, and J lie on the same line segment. Point F may lie on line $\overline{IJ}$ as shown, or if it is very slanted because of the constraints, or if points I or J are moved to the other side of the centerline, F may be on $\overline{IG}$ or $\overline{JH}$. The inverted form of the v-fold points up (at point B) rather than down, and the center fold is a valley fold instead of a mountain fold. Since $\angle ABC$ cannot be 180°, the point up and point down forms cannot be changed into one another, and therefore must be added separately, another reason for having one be the inverted form.

5.4.3.5 The Tent Element

The tent (Figure 5.14) is a 180° parallel element. It is constructed with an extra piece glued on each side of either a mountain fold, a valley fold, or a seam. The extra piece has a mountain center fold. Its constraints are quite similar to the step, with added constraints related to the fact that it has an added piece.

In order to fold properly, the basic constraints are:

- $\overline{AB}||\overline{CD}||\overline{EF}$
- $\overline{GH}||\overline{IJ}||\overline{KL}$
- The distance from $\overline{AB}$ to $\overline{EF} \leq$ the distance from $\overline{GH}$ to $\overline{KL}$. This prevents the tent from stopping the motion of the parent fold before it reaches 180°.
- The distance from $\overline{AB}$ to $\overline{CD}$ + the distance from $\overline{GH}$ to $\overline{IJ}$ = the distance from $\overline{EF}$ to $\overline{CD}$ + the distance from $\overline{KL}$ to $\overline{IJ}$.

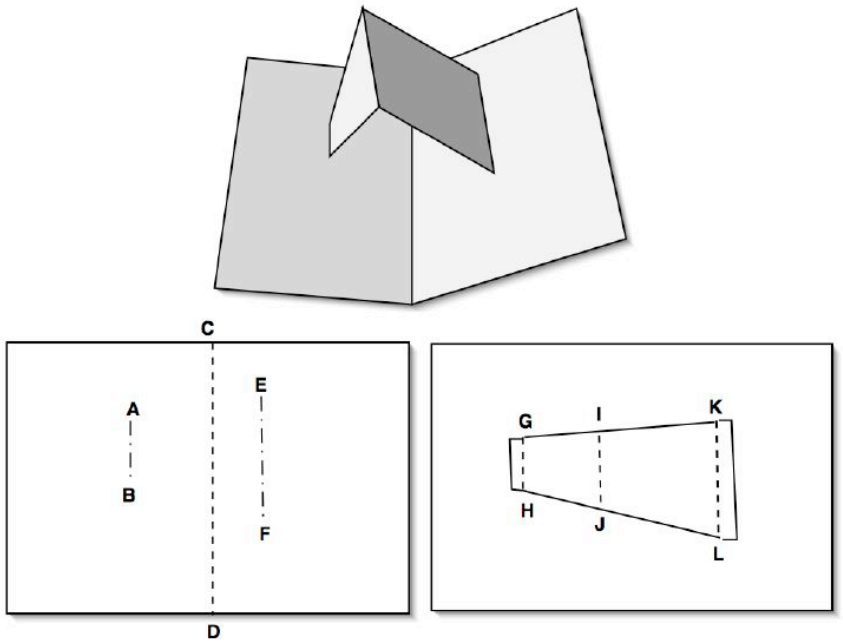


Figure 5.14: The Tent in Popup Workshop, showing the points important for the constraints.

- $AB = GH$ and $KL = EF$. That is, the seams on each piece must be the same length.
- The seams must also be offset in the same way. That is, (y-coordinate of point A) - (y-coordinate of point E) = (y-coordinate of point G) - (the y-coordinate of point K).
- Points A and B must lie on the left base plane. Points E and F must lie on the right base plane.

When an element of this type is added in Popup Workshop the starting conditions are:

- ABFE is a rectangle. Because of the constraints, GHLK is also a rectangle.
- The distance between $\overline{AB}$ and $\overline{CD}$ = the distance between $\overline{EF}$ and $\overline{CD}$.
- The distance between $\overline{GH}$ and $\overline{KL}$ is 1.5 times the distance between $\overline{AB}$ and $\overline{EF}$. If this causes the extra piece to extend beyond page boundaries, it is shortened to fit.

- The piece on the extra page is centered on the page horizontally and vertically in order to give the user room for changes.

For the purposes of Popup Workshop, points G, I, and K lie on a line. In addition, H, J, and L lie on a line. There is no inverted tent.

5.4.4 Constraint Methods in the Opening Algorithm

From the previous research on simulation of pop-ups discussed in Section 4.2, it is apparent that the most difficult single problem in designing tools for the paper engineer is the calculation of the positions of the elements during opening and closing of the page. Several methods have been proposed for this. Lee, Tor and Soo [66] suggest calculating the angles of the sides of the element, but limit their discussion to only two types of 180° elements, and develop different methods for calculating angled and parallel forms. Mitani and Suzuki [75], working with the limited subset of 90° elements used in origamic architecture are lucky in that a one-to-one mapping exists between the 3D and 2D forms, and a simple trigonometric relationship allows an easy calculation for any point. Glassner's [36, 37], solution utilizes the intersection of 3 spheres, leading to a complex, but more generalized solution.

In Popup Workshop a different approach to the problem has been implemented: a constraint system approach. Glassner identifies the possibility of using a constraint solving algorithm, then dismisses it:

> Constraint systems are flexible tools for solving complex problems. But they have three big drawbacks for this application: they are typically large and difficult to debug, they are notoriously sensitive to numerical instability, and they can get stuck while searching for a solution and end up with no solution at all. [36, p. 82]

To the contrary, in Popup Workshop, a constraint system has been found to work well. There are several reasons for this. First, it is not necessary to locate the opened position for a given point on the pop-up to a great degree of precision. The Viewer Window is not large, the user needs

only a rough idea of the what the result will look like, the positions are rounded to integer pixel locations during the display process, and finding the point to a rough approximation produces an animation that is good enough. Second, this is a situation in which the physical reality of the paper guarantees that there will be a solution. (In fact, there are always two solutions, as will be shown, for a given element in a given position and this does cause some difficulties.) Third, the algorithm for finding a given point is small and has been found easy to debug. Since every point is found with this algorithm, the entire process uses less code than considering the special cases that might be produced by each element. Finally, and related to the previous point, new element types can be easily added. There is no need to find the mathematical relationships between the points, as the same method can be used.

The Structure objects (each representing an element) are arranged in a linked list. Since they are added in order of their addition to the pop-up, there is a guarantee that when opening and element all of the elements below the current element are already opened. The opening algorithm progresses through the list in order, recalculating the point positions in each Structure before moving to the next.

The base page is a special case as it is assumed that the spine of the book remains fixed in position and the left and right pages move by the same amount. Therefore, when the page is fully opened the points assume their original values. In addition, points on the gutter do not change as the page is opened, so the x-, y- and z-coordinates of the other points are found by using the relationships:

- $\theta =$ angle of page opening in radians
- $a = x$ coordinate of the gutter
- $b = z$ coordinate of the gutter
- $c = x$ distance from the point to the gutter, perpendicular to the gutter
- $x = a - |(c * sin(\theta/2)|$ (if the point is on the left side of the gutter)

- $x = a + |(c * sin(\theta/2)|$ (if the point is on the right side of the gutter)
- y is unchanged
- $z = b + |(a * cos(theta/2)|$

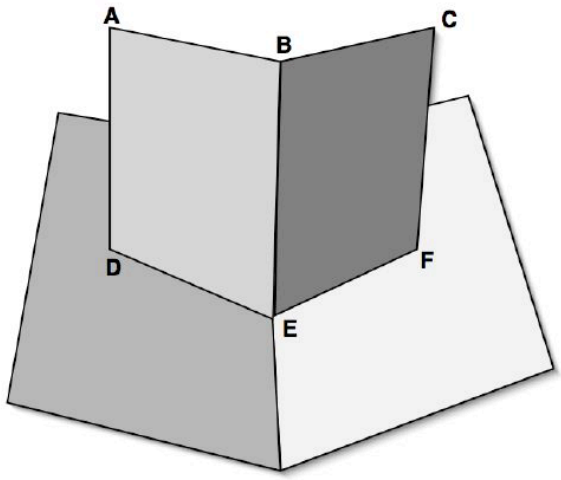


Figure 5.15: Example of Anchored and Unanchored Points: In this v-fold, A, B, and C are unanchored, and D, E, and F are anchored. The distances from A to B, D, and E, C to B, E, and F, and B to D, E, and F are always constant.

Each element consists of *anchored points*, that are parts of the planes beneath the elements that have already been opened, and *unanchored points*, that are not part of the underlying planes. Since the paper is assumed to be rigid, each unanchored point on a plane must remain at the same distance from the anchored points on that plane. For instance, in Figure 5.15 points A, B, and C of the v-fold are unanchored and points D, E, and F are anchored. If D, E, and F are known, A, B, and C may be found from the fact that distances on the planes do not change. B may be found from the fact that it remains at the same distance from D, E, and F at all times. In the same way, A remains at a constant distance from D, E, and B, and C does likewise from E, F, and B.

As a matter of fact, the same method can be used to find the positions of D, E, and F by using their distances from the corners of the plane on which they sit. Finding the positions of all

the points in an element is done by first locating the anchored point positions, and then using those points to locate the unanchored point positions.

```
Given:  thisPoint              //starting value (guess)
        pointA, pointB, pointC     //three points that we need to position it to
        distA, distB, distC        //distances that we want thisPoint to be from
                           //points A, B, and C respecively
        precision              // how close we think is close enough
        totalRepeats             //when do we want to say we've failed

Do:     Set newPoint to thisPoint
        Set newDistA to distance of newPoint from pointA,
           newDistB to distance of newPoint from pointB,
           newDistC to distance of newPoint from pointC,
        Set distAOff to abs(distA - newDistA)
           distBOff to abs(distB - newDistB)
           distCOff to abs(distC - newDistC)
        for(0 to totalRepeats)
           if(distAOff, distBOff, and distCOff are within precision)
              return newPoint
           else
              pick largest of distAOff, distBOff, distCOff and do:
                 make a line from newPoint to corresponding pointA, B, or C
                 move newPoint until the distance is correct (distA, B, or C)
              recalculate newDistA, newDistB, newDistC for changed newPoint
              recalculate distAOff, distBOff, distCOff
        return newPoint  //the best we could do
```

Figure 5.16: Algorithm for Placing Opened Pop-up Points (Pseudo-code): The simple, greedy algorithm used for each point in an element. The algorithm uses triangulation and iterates until the distances to the new point match to the precision desired, or the number of iterations passes a limiting value. At each step, the worst fit distance is adjusted to the correct value.

The problem with finding the positions of the points on an element, therefore, always resolves to one of finding a new point such that the original distance from that point to three other points remains the same when the three points are moved. The open method in the element itself can take into account which points are anchored, and which points should be used to find the unanchored points.

A simple routine is used for finding the position of all points in any element. The first algorithm tried (in version 1.0) used random hill-climbing. This was found to converge slowly, and was replaced in version 1.1 by the current greedy algorithm shown in Figure 5.16. Briefly, if the point is not at the correct distance from all three points, it is moved away or toward the worst fitting point until that distance is correct.

When opening and closing the pop-up in the Viewer Window, the starting position of a

point is usually the last position of the point, as it seldom moves far from its last position as the position of the slider control is passed to the program frequently. If the element is being opened for the first time, a position is chosen heuristically that is at least in the vicinity of where it should be. For instance, in the case of the v-fold in Figure 5.15, the starting position for B is chosen with the same x- and y-coordinates as E, but with a z-coordinate equal to the distance EF plus the z-coordinate of E, in order to set the starting position out from the base planes in the correct z direction. It has been found that the starting position is important, as it influences the speed with which the algorithm converges on the correct location.

The algorithm is sufficiently fast, and in user tests speed was not a problem. 90° elements in particular converge quickly, and most points are found within 50 iterations when opening and closing a step element. V-folds and tents are somewhat slower, perhaps because their movement is more extreme, especially in the case of v-folds. The animation of these elements can exceed the iteration limit (currently 5000) but still produce an acceptable view of the pop-up. Additional refinement of the starting point heuristics and adjustment of the precision and iteration limits could improve this performance.

The only current problems arise because there are two stable states for any element. For 90° elements, these are the usual opened configuration and the case in which the element does not “pop out”, that is, it remains as part of the original page. This seldom causes a problem, as this is easily recognized (the unanchored points are in the same plane as the anchored points) and corrected. Occasionally, if the 90° element is on a v-fold that takes it far out of the original planes of the page, this will become an issue. V-folds and tents, on the other hand, have a stable state that is on the opposite side of the page from the desired state, as if the element were attached to the back of the page or supporting elements, and this can occasionally be the state found. These aberrations were not a problem in user testing as they occurred infrequently, and the children were amused rather than confused by the resulting configuration.

There are several ways in which these problems might be cured. One way, for example, would be to transform the anchored points to a position that makes the base for the element

oriented similarly to the base page, and to check whether the found points were on the correct side of the anchored points. Another solution could be to find both positions and disambiguate in some manner later. This problem has occurred for other researchers. Glassner, for example, keeps track of the side on which the element sits, as his intersection of spheres method also finds two possible points for the tip of a v-fold. These display problems have no effect on the function of the completed pop-up as the pattern can still be constructed correctly.

5.5 Summary

The Popup Workshop interface consists of one or more Editor Windows and one Viewer Window. The Editor Window is where the pattern is drawn using the mouse, and from where the pattern is printed for cutting and folding. Five buttons in the Element Tool Palette are used for selecting which element to draw: step, beak, angled step, v-fold or tent. Once the element type is selected, the user adds an element by placing the cursor near a fold or seam and then clicking and dragging. Once elements are added, they may be changed, deleted, or replicated onto matching folds of the element beneath them. There are also simple decoration tools that create filled planes and drawn lines. The Viewer Window shows a 3-dimensional representation of the element that can be rotated and opened or closed. Menu items allow printing the pattern for construction, exporting a JPEG version of the pattern, starting a new blank pop-up, saving the pop-up for reuse in Popup Workshop or opening a previously saved version.

Popup Workshop was written in Java, primarily for portability. The class structure, however arose naturally due to the modularity of pop-ups. The basic unit is a structure, and structures (elements) are kept in a linked list in the order in which they are added. This allows all elements to be processed for new locations when the opening angle is changed by simply traversing the structure list. The methods for changing and deleting elements are recursive to allow any elements above the changed or deleted element to be changed or deleted as well.

There are two uses of constraints in Popup Workshop. First are the geometric constraints that apply all elements. These are enforced during the construction of an element and all changes

to it, so barring collisions between elements the pop-up will properly fold. In the Viewer Window constraint processing is used to calculate new point locations when opening and closing pop-ups, since distances along a plane between points do not change. This allows the calculation of new point locations for each element to act in the same way.

In Chapter 6, the user testing activities for Popup Workshop will be described, including discussions about the users, the methods, and the results. Assessment of the results will be based on the previously introduced craft framework of knowledge, skill, and appreciation.