

Chapter Twelve

DEFINING PATHS

In Proof animations, motion most commonly occurs along fixed routes, such as roads, aisles, railroad tracks, etc. We call such motion *guided* motion. The ability to handle guided motion easily is one of Proof's most important capabilities.

In Chapter 7, we explored the use of Paths that had already been defined. In this chapter we explain how such Paths are defined.

Paths are superimposed upon existing Lines and Arcs. Thus, Paths depend on the presence of underlying geometry. You must first draw (in Draw Mode) or import the geometry atop which Paths are to be defined. Once you are satisfied with the geometry of your layout, you can begin defining the Paths for that layout.

About Path Mode

Proof provides a special Path definition mode that you can select by choosing **Mode, Path**. When you enter Path Mode, your drawing takes on a different appearance. Paths, and only Paths, are shown in *color*. The *other layout elements* are displayed in *black on a gray background*.

The first time you invoke Path Mode with a new layout, you will not see any colors, because no Paths have yet been defined. Before proceeding, you may find it useful to examine one of the layouts you have already used, such as `path.lay` or `apath.lay`, to see what a layout containing Paths looks like in Path Mode.

In a layout with *existing* Paths, most Paths are shown in *light blue*. Any Paths that are in need of *repairs* (discussed later in this chapter) are shown in *white*.

The Path currently selected for editing (if any) is shown in yellow, and the selected Segment (defined later in this chapter) of the currently selected Path is displayed in red.

You cannot change Proof's Path Mode color scheme.

The Path Mode Toolbar

Entering Path Mode causes the Path Mode toolbar to appear:



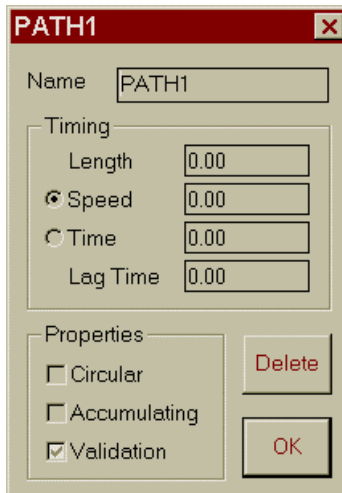
Figure 12-1. The Path Mode Toolbar

The tools on the Path Mode toolbar are described below.

Adding a New Path

When in Path Mode, you use the **+P Add Path** button to add a new Path, or choose **Path, New**. Proof will prompt you to supply a unique name for the new Path. Path names, like other names in Proof are case sensitive, can contain uppercase letters, lowercase letters, and/or digits, must begin with a letter, and are of unrestricted length.

Once you have typed in the Path name and clicked **OK** or pressed **<Enter>**, Proof will display a dialog box for specifying the Path's properties:



If you wish to begin adding Segments to the new Path at this point, note that you will be in “Insert After” mode. Insertion modes are discussed in detail below.

Naming Conventions for a Network of Paths

In a network of Paths that will be driven by a parameterized section of code in a simulation model, it may be convenient to adopt a convention of naming the Path with a *base name* plus a *number*. For example, the Paths in the KANBAN model (used in Exercise 12-4 at the end of this chapter) are named CP1, CP2, CP3,..., CP32. With these names, the model can refer to a particular Path in this set of Paths through a stored numeric data value. In other words, although the concept of a “set of Paths” is not explicitly implemented in Proof, you can apply your own naming conventions to get a similar effect. See Chapter 17 for more discussion of networks of Paths.

Renaming a Path

If you would like to change the name of the currently active Path, simply select the **Name** control in the Path dialog box and type in a new name.

Selecting an Existing Path

In Path Mode there are two ways to select an existing Path for examination, editing, or deletion. The first is to click on or near the Path in question. If multiple Paths share the Line or Arc that you happen to click near, subsequent clicks will select the other Paths in sequence.

The other way to select a Path is by choosing **Path, Select**, which allows you to choose the Path’s name from a list of all defined Paths in the layout. If you should ever accidentally define an empty Path – one with no Segments – this is your only option for re-editing or deleting the Path.

Each Path has a label (the Path’s name) that is displayed on the screen in Path Mode. Although label locations have no bearing on the animation, you can drag a selected Path’s label to a different place with the mouse if that will help you see things more clearly in Path Mode. The position of each label is stored in the layout file.

The selected Path is always highlighted in yellow and red.

Completing a Path

When you are finished working on a particular Path, you can close the Path dialog box or click on another menu item or toolbar icon to move into another operation. There is no “save path” button.

*New Paths or Path modifications always become part of your layout immediately, but they will not be saved to disk until you select **Save Layout** from the **File** menu.*

If you change Modes while working on a Path, then return to Path Mode during the same Proof session, the selected Path and Segment will remain selected.

Deleting a Path

To delete a Path, you must first select it and then click on the **Delete** button. Proof will prompt you for verification. If you click **Yes**, the selected (highlighted) Path will be deleted, and you can now re-use the Path name if you choose.

There is no Undo for Path deletion.

Path Properties

In the sections that follow, we’ll discuss Path properties and how you specify them.

Length

Proof automatically displays the **Length** of the current Path in Proof coordinate units. This is the distance along the Path from the beginning of the first Segment to the end of the last Segment. You cannot override the Length of a Path.

Speed

Path **Speed** is used as the speed of all Objects placed on a Path, unless specifically overridden for individual Objects or Classes of Objects. (See Chapter 14 for ways an Object’s speed can override the Path speed.) If you specify a **Speed**, the radio button to the left of the word **Speed** will be selected. This indicates that the Path’s Time (travel time; see Time below) is calculated and displayed automatically based on the specified speed and the calculated length. For example, if the Length is 75.0 and you specify a **Speed** of 5, then the Time will be 15.0. If you specify a **Speed**, it will carry over to each subsequently defined Path until you designate a different **Speed** or a **Time**. This is convenient if you need to define several similar Paths with

the same Speed.

Time

Time (travel time) is the amount of time an Object will take to travel the Path from the beginning of the first Segment to the end of the last Segment. If you specify a **Time**, then the radio button to the left of the word **Time** will be selected. In this case, Speed (see Speed above) is calculated and displayed automatically based on the specified **Time** and the calculated Length. For example, if the Length is 75.0 and you specify a **Time** of 150, the Speed control will be 0.5.

If you specify a Time, it will carry over to each subsequently defined Path until you designate a different Speed or a Time. This is convenient if you need to define several similar Paths with the same travel time.

Accumulating Paths

Proof supports two kinds of Paths, **Accumulating** and **Non-accumulating**. Accumulating Paths are used for animating Path motion in which Objects need to queue up at the end of a Path. For example, an Accumulating Path could be used to show customers lining up for service at a bank. Accumulation is turned on or off by means of the check box to the left of the word **Accumulating**. The current setting carries forward to each subsequently defined Path. By default, accumulation is disabled (not checked).

If an Object travels toward the end of an Accumulating Path, it will normally “accumulate” visibly, behind other Objects already resting at the end of that Path. But in order for accumulation to take place, the Objects must have non-zero clearances (see Chapter 11).

An Object that reaches the end of an Accumulating Path (without encountering another Object in front of it) will come to rest with its Hot Point on the end of the last Segment of the Path. The next arriving Object will stop with its Hot Point behind the Hot Point of the first Object by an amount equal to the *aft* clearance of the “resting” Object ahead plus the arriving Object’s own *fore* clearance (see Figure 12-2). Additional arriving Objects are handled similarly.

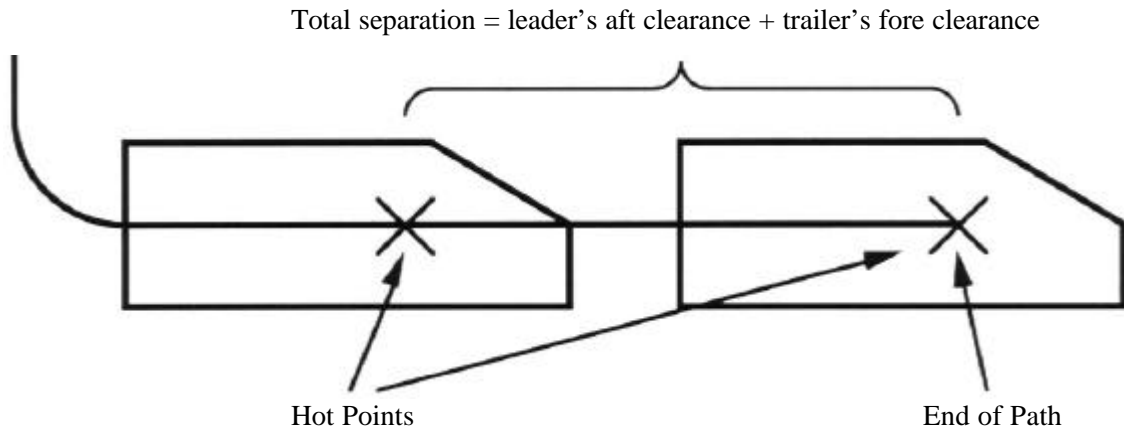


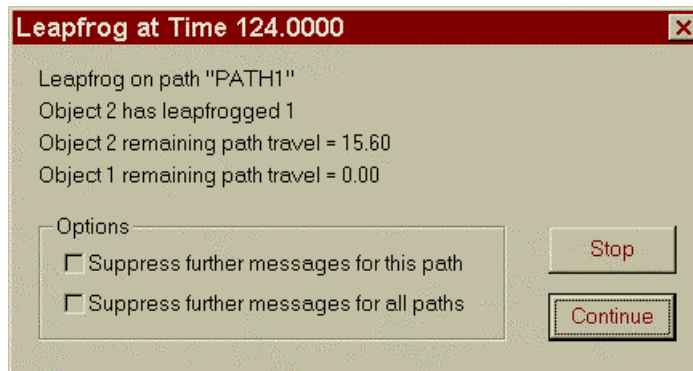
Figure 12-2. Two Objects at the end of an Accumulating Path

When the first Object is removed from the Accumulating Path, the other Objects will move forward automatically. An Accumulating Path cannot be circular. Circular Paths are described below

Validation

During animation, Proof produces alerts when exceptions occur in the use of Accumulating Paths. There are two kinds of these exceptions. *Leapfrog* alerts occur when an Object other than the first Object is removed from an Accumulating Path. *Encroachment* alerts occur when an Object is placed on an Accumulating Path in a manner that violates fore/aft Object clearances you have specified.

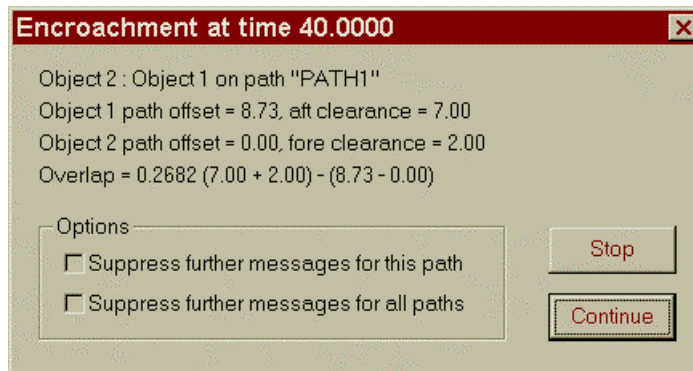
Recognition of **Leapfrog** errors is very helpful in detecting modeling errors in which a FIFO queue discipline (first-in-first-out) is violated. Most physical queueing systems are FIFO. When a Leapfrog error occurs, an "OUCH!" designation is displayed at the point of removal, and the following alert dialog appears:



In the case shown above, Object 2, which has a remaining Path travel of 9.0, has leapfrogged Object 1, which has a remaining Path travel of zero. Note that the dialog box gives you a number of temporary options for dealing with Leapfrog errors as the current animation progresses. If you're in the initial stages of debugging an animation, you may choose to suppress additional leapfrog messages. Doing so will allow you to proceed further without being harangued by an incessant flow of alerts. If leapfrogs are a normal occurrence on a Path, you may permanently turn off leapfrog recognition. We'll describe how to do this in the section named Validation, below.

Encroachment Alerts arise when an Object is placed on an accumulating Path before there is sufficient room on the Path (e.g., a conveyor speed is too slow to handle incoming traffic). "Sufficient room" is determined by comparing the linear separation of the Objects' respective Hot Points with the accumulation separation (aft clearance of leading Object plus fore clearance of trailing Object).

Usually an Encroachment Alert indicates a modeling or system design deficiency. Sometimes it only means that your Object clearances are incorrect. When an Encroachment Alert is issued, the point at which the error has occurred is shown with an "OUCH!" message, and a dialog box similar to the following appears:



In the above example, Object 2 has encroached on Object 1. The distance between the Hot Points of the two objects is 8.73. (Remember, an Object's location on a Path is determined by the Object's Hot Point.) The sum of the aft clearance of Object 1 and the fore clearance of Object 2 is 9 (7+2). Since a minimum separation of 9 is required, and Object 2 has been placed on PATH1 in such a way that the actual separation is 8.73, an Encroachment error occurs.

The **Validation** check-box (in the Path dialog) controls, on a path-by-path basis, whether Proof issues alerts for Leapfrog and Encroachment errors for a given Path. This property is saved for each Path when you save your layout. Note that Validation is available only for Accumulating Paths.

If Validation is enabled, and you get Leapfrog or Encroachment Alerts that you do not understand, you need to ascertain why the errors are occurring. Do not ignore the Alerts or disable Validation unless you understand the problem. These alerts were built into Proof for good reason: they almost always indicate modeling errors. Remember that you can look in the .LOG file for a detailed report of such errors.

Lag Time

Lag Time is a path-specific delay that Proof enforces before a stopped Object is allowed to resume its motion along an Accumulating Path. Lag Time is zero by default. You can set it to a non-zero value if you want to show slippage or approximate acceleration time for independently moving Objects.

An example of Lag Time use is a Path containing automobiles stopped at a traffic signal. When the signal turns green, each car waits until shortly after the car in front of it begins moving.

You cannot specify a Lag Time unless the Path is an Accumulating Path. While you remain in

Path Mode, the most recent Lag Time setting carries forward to each subsequently defined Path until you change it.

Circularity

Objects placed on a Circular Path circulate until removed from the Path. Objects that reach the end of the Path proceed directly to the beginning of the Path even if the end is not connected to the beginning.

Circularity is enabled/disabled by clicking on **Circular** in the Path dialog box. Since few Paths are circular, **Circular** is disabled by default. However, the current setting carries forward to each subsequently defined Path.

The use of Circular Paths is commonplace (although not required) in animating material handling systems. For example, factories frequently contain overhead conveyor systems in which a chain circulates around the factory, with carriers suspended from the chain. Circular Paths are explored later in this chapter.

About Path Segments

All usable Proof Paths consist of one or more *Segments*. Each Segment is superimposed over all or part of a single Line or Arc. You create a Path graphically, specifying the Segments by clicking on Lines and Arcs. Each endpoint of a Segment is always either (1) the endpoint of a Line or Arc, or (2) an intersection of Lines and/or Arcs.

The concept of individual Segments has meaning only in Path Mode, when you are defining or editing a Path.

For editing purposes, the Segments of a Path are sequentially ordered. Every Path has a first and last Segment. Each Segment has a direction. The directions of Segments are indicated on your screen by arrows at the “downstream” end (head) of each Segment. The “upstream” end (tail) is unmarked. These arrows show *only* when you are in Path Mode – they do not show in Draw or Run Modes. But collectively the Segment directions do determine the direction of motion for Objects placed on the Path.

Consecutive Segments of a Path are usually connected at their end points, but it is possible to have a *disjoint* Path in which there are one or more apparent gaps between Segments. Objects moving along such a Path will appear to jump over these gaps. Only the actual Segments are part of the Path.



*Segments are not individually addressable from the animation trace stream. If you have the urge to place Objects on individual Path Segments, you should use more (shorter) Paths to achieve the needed degree of control, although the **place on...at** command is also at your disposal (see Chapter 14). The topic of “long” vs. “short” Paths is also discussed further at the end of this chapter.*

Adding and Deleting Segments



Insertion Modes

Path Mode offers two ways of adding Segments to a Path: the **Insert After** mode and the **Insert Before** mode. In Insert Aftermode, each new Segment (or collection of Segments) is placed beyond the downstream end of the current Path Segment. In Insert Before mode, each new Segment (or collection of Segments) is placed before the upstream end of the current Path Segment.

Normally, Segments are added to a Path in Insert After mode.

Insert After mode is selected by clicking the  button, and in Insert Before mode is selected by clicking the  button. Once you enter into Insert After or Insert Before mode, the respective button will remain “depressed” in the toolbar, providing a visual indication of the current mode of operation.

Getting Started

When you create a new Path, you are automatically placed in the Insert After mode. To define the very first Segment of a new Path, click near the downstream end of portion of a Line or Arc. As you do so, the Line or Arc changes to red, indicating that you’ve created a Segment and that this is the active Segment. If you click on the wrong end of the right piece of geometry, you can reverse the first Segment’s direction by clicking on the  **Reverse Segment** button, and click on  **Insert After** to resume Insert After mode. If there are other Lines or Arcs intersecting the Line or Arc on which you want to start your Path, your first click should be near the downstream portion of the first *Segment* of the Line or Arc, i.e. before the first intersection point but closer to that point than to the intended beginning of the Path.

Inserting Multiple Segments with a Single Click

It is not necessary to click on each and every piece of Line and Arc comprising a Path. Proof has a path-finding algorithm that enables it to quickly find the shortest route between the current


end of a Path and the next place in the geometry where you have clicked. Proof will automatically create and display the Segments for you. If Proof cannot find a continuous route between the current end of a Path and where you have clicked, it will ask you if you want a Disjoint Segment. Generally you will click “no” and try again.

It is legitimate for Segments of a Path to be Disjoint (not connected to one another). However, the built-in automatic connection feature makes it difficult to specify a Disjoint Segment if there are feasible connections. If you really want to create a Disjoint Path in this case, you usually must delete one or more Segments that have already been inserted.

Once you get used to Proof’s long distance algorithms for multiple Segment insertion, you will find this to be a time-saving feature. However, if you are not used to the algorithm, you might prefer to add Path Segments one right after another by always clicking on the adjacent section of Line or Arc.

Verifying Path Correctness

Proof offers several tools for verifying Path correctness. One of the most useful is the

 **Check Path** button. When you click on this button, Proof will cycle through the segments comprising the active Path, highlighting each one in red, in sequence. If you’ve accidentally caused a Path to double back over itself, you’ll quickly see your error when Proof travels the Path.

You can manually step through the segments of a Path by using the following buttons:



 Select the **First** Path Segment

 Select the **Previous** Path Segment

 Select the **Next** Path Segment

 Select the **Last** Path Segment

Correcting Mistakes

If you make a mistake, you can click on the  **Delete Segment** button to delete the current (red-colored) Segment. Click repeatedly to retract Path Segments added in error. Alternatively, if you’ve made a *big* mistake with a single click, use the  **Undo** button. After you’ve gotten rid of the erroneous Segments, you can resume adding Segments by clicking the

 **Insert After** or  **Insert Before** button.

Repairing a Corrupted Path

If the geometry of one or more of the Lines or Arcs underneath a Path is modified in Draw Mode in a way that would change the Path (see below), you will need to repair the Path. If one or more Paths have been *corrupted* in this way, the **Repair Damage** menu item in Path Mode's **Path** pop-up menu will be activated. Normally this item is grayed out. If you click on this menu item, Proof will present you with a list of Paths requiring repair.

About Corrupted Paths

If in using Draw Mode, you alter the elements that underlie a Path definition, that Path can become corrupted. A corrupted Path may be unusable in a subsequent animation.

Here is a list of ways a Path can become corrupted:

- A Line or Arc underlying the Path is deleted
- A Line or Arc is edited in a way that moves or changes the part underlying the Path

Here is a list of modifications that *will not* corrupt a Path:

- A Line or Arc underlying the Path changes to a different color
- A Line or Arc is changed through a Box Edit operation (scale, translate, rotate) that includes all of the elements underlying the Path
- A Line or Arc is edited in a way that does not move or change the part underlying the Path

If you accidentally do something while in Draw Mode that corrupts one or more Paths, you will immediately see an Alert that says, "You have just corrupted path xxx; consider UNDO!" If in doubt, **Undo** what you did! *This is the best way to deal with corrupted Paths, because it is much easier than repairing the Paths later in Path Mode.*

You can also correct misplaced or deleted Lines and Arcs by hand to reverse the corruption *if you do so before leaving Draw Mode and before saving the layout*. This works even if it is too late for Undo. Proof will consider a Path repaired if it sees that the edited geometry matches the geometry underlying the original Path. This is called "automatic repair."

If you corrupt a Path and then switch to Path Mode, it is too late for automatic repair. At this point you can either revert to your previously saved version of the layout (use the File menu to open the layout file, or exit and restart Proof), or you can use the Repairs facility in Path Mode to identify the corrupted Path for manual reconstruction.

If you corrupt a Path and then save the layout, it is also too late for automatic repair. At this

point you can either revert to your previously saved version of the layout (stored in a .ZZZ file), or, if the layout is still open, you can use the Repairs facility in Path Mode to identify the corrupted Path for manual reconstruction.

Corrupted Path Example

In Figure 12-3 “Path 1” is comprised of Segments AB, BC and CD. The Path Segment BC is part of the Line EF.

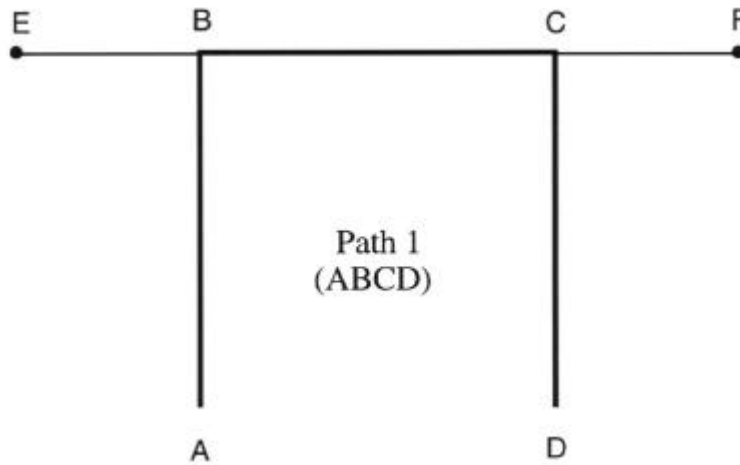


Figure 12-3. Path 1 is comprised of Segments AB, BC and CD

Proof “knows” that if the layout is changed in Draw Mode so that a Line no longer passes through both points B and C, Path 1 will become corrupted. Here are two examples of how this could occur:

- Line EF is deleted
- Line EF is shortened or moved vertically, so it no longer intersects point(s) B and/or C

However, if Line EF is extended horizontally, the Path will not be corrupted. If Line EF is removed but then replaced by another Line that touches points B and C, and the layout has not been saved and Path Mode has not been entered, then the Path will be automatically repaired.

Using Repairs

Corrupted Paths show up in Path Mode as *white* instead of blue. (Exception: the one Path that is selected, if any, will have red and yellow Segments regardless of whether or not it is corrupted.)

When you click **Path, Repair Damage**, an alphabetized list of all corrupted Paths appears. A Path will stay on this “repairs list” until you confirm that it has been repaired, or until you save and re-open the file. When the list of corrupted Paths becomes empty, the **Repair Damage** menu item will revert to a grayed-out state. When you click on the Path name in the repairs list (or type the name into the dialog box), that Path is made active for normal Path editing. The usual toolbox menu lets you edit the active Path’s properties and Segments.

You may have to go back into Draw Mode to add or edit the underlying Lines and Arcs before you can repair the Path Segments to your satisfaction.

“Should path *pathname* be removed from the repairs list?”

You will always see “Should path *pathname* be removed from the repairs list?” as soon as you indicate that you are finished working on any Path that is on the repairs list. *Clicking “Yes” in response to this prompt is the only way you can remove a Path from the Repairs list*, aside from saving and re-opening the layout (or deleting the Path). Proof needs you to verify your manual “un-corruption” of each corrupted Path.

Actions that indicate you are finished with the current Path include adding a new Path, selecting another Path, saving the layout, moving to another Mode, or clicking **Repairs**.

Sometimes you might see “Should path *pathname* be removed from the repairs list?” even though you may have not clicked Repairs. For example, you might go from Path Mode to Draw Mode, corrupt the Path that is still selected in Path Mode, then return to Path Mode. Or, you might select a corrupted Path by choosing **Select Path** instead of through **Repairs**. In any of these cases, you are working on a corrupted Path, so you will be prompted when you finish working on it.

Managing Disjoint Segments: What Counts as an Intersection?

It is possible that Lines and Arcs may *appear* to intersect when they really don’t under strict geometric interpretation. For example, if line 1 ends at (10,10), and line 2 starts at (10,10.00001), the lines will appear to intersect on the screen, because the end of line 1 and the start of line 2 will round to the same pixel. Under strict geometric interpretation, the two lines obviously do not intersect. This problem is particularly prevalent in imported CAD drawings,

which, because they are usually drawn only to visual scrutiny, and are frequently less precise than Proof's route-finding algorithm expects.

Layouts drawn in Proof “by eye,” rather than using Proof's “Snap-to” features, will almost always contain apparent intersections that aren't true intersections.

When determining what counts as a *true* intersection, Proof uses a so-called “Visual Epsilon” criterion. According to this criterion, a potential intersection counts as an intersection if (1) it's a true intersection, or (2) any algebraic gap is smaller than a single pixel at the current viewing scale. In other words, if you can see a gap, it counts; otherwise, it doesn't.

What counts as a true intersection in Path Mode depends on the current viewing scale unless you have adjusted the Tolerance value (see below).

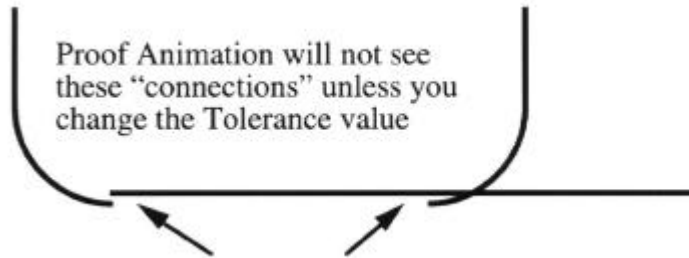


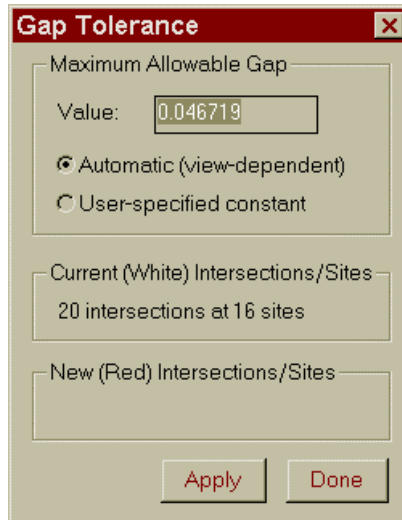
Figure 12-4. Disjoint Segments

Troubles Arising from Gaps in a Drawing

You may click at a given point in a drawing and expect that Proof will extend the current Path using the “obviously” shortest route but encounter a problem due to a **Gap**. Gaps can cause two types of unexpected behavior when you are adding Segments to a Path. First, if there's a gap along the “obvious” route, Proof will try very hard to find an alternative route. Such routes can be astonishingly circuitous and not “obvious” at all. If this happens to you, you'll be grateful for the Undo button. A second problem that can occur is that gaps may prevent Proof from finding any route at all from the previous end of the Path to where you have clicked. If this happens, Proof will tell you that no route can be found and ask you whether it's **OK** to create a **Disjoint Path**.

Dealing with Gaps Effectively

You can often verify the presence of gaps by simply zooming in on a Path. For deeper problems you can explore *gaps* in detail by choosing **Setup, Intersection Gap Tolerance**, and the following dialog box appears:



The Value shown is the current **Gap Tolerance**. All gaps smaller than this value are treated as if they were intersections. All intersections currently recognized under this criterion will be marked with white “X” indicators. If you select the **User-specified constant** option, you will be allowed to enter your own gap tolerance value. If you do so, intersections recognized under the new criterion will be indicated by red dots, and red boxes will be drawn around the intersections, indicating the area encompassed by your choice of gap tolerance.

If you choose too small a value for gap tolerance, true intersections may be rejected, due to computational roundoff errors. If you choose too large a value, unintended intersections will be recognized. Choosing gap tolerance correctly can be very tricky. For example, if you’re drawing a railroad with parallel tracks and lots of intersections, and you choose a gap tolerance that is larger than the distance between parallel tracks, you’ll end up creating paths that jump from track-to-track at junctions.

If you set a **User-specified** gap tolerance (**Maximum Allowable Gap**), your value will remain in effect until you exit Proof. Gap tolerance will be saved permanently if (and only if) you


update the Home View before saving your layout.

Exercise 12-1: Setting Gap Tolerance for Path Segments

Bring up the layout file named `toler.lay` in the exercise folder.

Now choose **Mode, Path** to get into Path Mode. You should see a Path with a single Segment called “swim1.”

Click on the Path or the label “swim1” and it should now be highlighted in red.

Let’s add another Segment to the Path. Click on the  **Insert After** button, then move the mouse to the vertical Line on the right. Click on that Line to add as another Segment.

What happens when you try to add another Segment? Click on **No**. Do the Lines look connected? Let’s take a closer look at the intersection of the Lines that contain the Segments. Use **Zoom Box** in the **View** menu to zoom in on that area.

On closer inspection, you can see that those Lines are not really connected. There are a couple of options available to correct the problem: one is to physically change the Lines in Draw Mode, or set a Tolerance limit that will overlook small gaps in “connected” Lines.

To experiment with alternative values of gap tolerance, click **Setup, Intersection Gap Tolerance**. Turn on the **User-specified constant** option, and type in a number[suggest a value here for the user] that will be large enough for your Segments to be considered connected. Now go back to Path Mode and try again.

Using Path Mode While an Animation is Running

You can enter Path Mode at any time to change Segments or properties of a Path. However, if you modify the Path Segments and try to resume running the animation, you will be advised via an Alert that you must **Save** and re-**Open** the layout in order to run the animation. Remember that you can reopen your current layout file and animation trace file by clicking on **File, Open Layout & Trace**.

Exercise 12-2: Defining and Using a Path

Bring up the `trimtest.lay` layout under Proof. (This file is located in the exercise folder.) Define a single Path of your own design using the existing Arcs and Lines. Put in whatever properties you would like. Then create a simple Object Class in Class Mode. Do not give the Objects a speed, but remember to give your Path a non-zero speed. Save your layout.

Using your text editor or word processor, create a simple trace file called `trimtest.atf`. Create multiple Objects from your Object Class and place the Objects on your Path. Be sure to separate your `place...on` commands with some `time` commands.

Bring up the `trimtest` animation (`trimtest.lay` & `trimtest.atf`). How does it look?

Exercise 12-3: Defining and Using a Path, Part 2

Bring up your `guide.lay` layout (from Exercise 10-3). (If you did not create your own `guide.lay` file, a completed version is located in the `sample` folder.)

Click **Mode, Path**. Define a single Path as `CircPath`, (select **Path, New**) starting and ending at the indicated point in Figure 12-5. Your Path does not have to be identical to the one in the figure. (Arrows on the curved Segments will show up in your Path. They are omitted from the figure.)

Make the Path's speed 20.

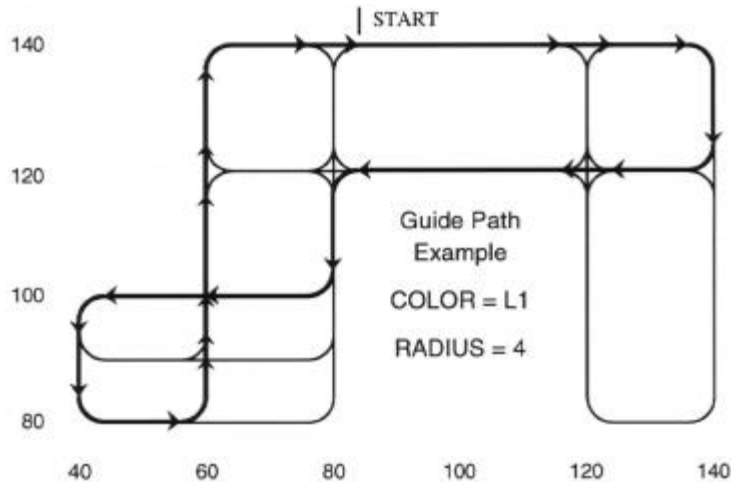


Figure 12-5. Guide Path example

Next define a directional Object Class of your choosing and then save `guide.lay`.

Finally, use your text editor or word processor to edit a new `guide.atf` file and put some Objects on your new Path at various times.

Bring up the `guide` animation (`guide.lay` & `guide.atf`).

Although you’ve defined one large Path, an animation of a guidepath model with intersections, such as this one, will typically contain many small Paths. Access to each Path is then controlled by the model.

Exercise 12-4: Exploring Paths in Sample Animations

Bring up the `kanban` animation, located in the `sample` folder. Look at some of the Paths in Path Mode. Which is the longest? What characteristics of a model make it suited to this style of Path usage?

Try the same experiment with the `newsprnt` animation in the `sample` folder.

Circular Paths

A Path can be **Circular**. You might have tried making your Path circular in Exercise 12-3. The Circular property means that Objects reaching the end of the Path will never stop, but will continue on immediately from the beginning. There are two principal uses of Circular Paths.

The main use of Circular Paths is for animating the physical re-circulation of vehicles or carriers. The `newsprnt` animation is such a system. Rolls of newsprint are transported on carriers which move continuously through the system. The main loop can be animated using a Circular Path.

Circular Paths are also useful in abstract representations of motion. For example, a small squiggly Circular Path inside a machine can be host to a rapidly moving Object, the only purpose of which is to represent a “machine busy” state.

Exercise 12-5: A Circular Path Example

Bring up the `newsprnt` animation, located in the `sample` folder, and run it. In this animation, empty “dogs” circulate continuously along a main Circular Path. When a dog picks up a load, it changes shape. (Changing an Object’s shape is covered in Chapter 13.)

Long Paths vs. Short Paths

When travel from point A to point B covers many pieces of Lines and/or Arcs, there are two choices for designing a Proof Path. One can construct a single, “long” Path, or one can construct a sequence of “short” Paths that are connected end-to-end. The best approach depends on the nature of the system being modeled and the manner in which the model of the system is constructed.

Long Paths have the advantages of (1) simplifying operations such as shutdowns of long-distance conveyors and (2) allowing automatic accumulation over long distances. If there are only one or a few main routes, as in the `newsprnt` example in the `sample` folder, long Paths make sense even if Objects are getting on and off at different points along the Paths..

One key disadvantage of long Paths is the number of Paths required when sources and destinations are each at the end of individual spurs along a long, “main” line. Full interconnection among all sources and all destinations requires a number of Paths equal to the number of sources times the number of destinations.

Short Paths are also easier to work with if the Path’s beginning and ending points can be conveniently placed on actual stop-start points. For example, conveyor or guided vehicle control logic frequently is based on the principle of zones, where at most one load or vehicle is permitted to occupy a zone at a given time. In such systems, it is probably best to use “short” Paths that correspond to zones used in the system’s control logic. This approach was used in the `kanban` simulation located in the `sample` folder.

A disadvantage of short Paths is that table-driven movement through a network of Paths may at first be more difficult to program than directly connected long Paths.