A Spatial Model for Nested Multiscale Interfaces

Chris Olston
Stanford University
olston@cs.stanford.edu

Abstract

Significant recent interest has been devoted to the multiscale interface as a paradigm for user-driven exploration of large or complex information worlds. Multiscale interfaces sometimes permit nesting of worlds, a notion that underpins a diverse variety of navigation aides and multiple-view tools such as visual hyperlinks, bookmarks, filters, magnifying glasses, overview and detail views, and coordinated views, all of which can be used to enhance the effectiveness of information exploration. These constructs often behave in accordance with underlying spatial relationships among nested views. Unfortunately, researchers and designers currently lack tools to help them describe and reason formally about spatial relationships tying together nested virtual worlds. The absence of appropriate formalisms is problematic because multiple spatial models are available and, even under a consistent model different nested constructs can exhibit different behaviors, many of which have complex and asymmetric properties. In this paper we describe a spatial model for nested multiscale interfaces that we constructed based on experience with a prototype data visualization system. We show how behaviors of nested components can be described succinctly in our model and provide a comprehensive taxonomy of 32 distinct behaviors. Finally, we make use of our spatial model to design techniques to help maintain user orientation during navigation between nested worlds by smoothly animating transitions between outer and inner views.

1 Introduction

Multiscale interfaces [FB95] represent a powerful and flexible paradigm for user-driven exploration of large information worlds in which users can adjust the scale with which information is displayed while browsing. Multiscale interfaces have been proposed or deployed in a diverse set of applications including information visualization tools, *e.g.*, [AW95, BHP$^+$96, DKR97, LRB$^+$97, STH02, WOA$^+$01], WYSIWYG tools for creating and viewing documents, *e.g.*, [Cora, Corb, MPBH95], geographic information system (GIS) and cartographic tools, *e.g.*, [AA99, BGS99], and even Web browsers [BHS$^+$96]. Many multiscale interface environments including [ACSW96, BHP$^+$96, BHS$^+$96, BSP$^+$93, PF93, PM99, SFB94, WOA$^+$01] permit nesting, whereby one information world appears as a sub-window of another. Nested sub-windows are called *portals* [BHP$^+$96, WOA$^+$01] [1], which serve as a unified way to implement a number of useful constructs that can enhance the ability of users to explore large or complex information worlds effectively. First of all, portals provide a convenient mechanism for linking together different information worlds or different locations within the same world via a useful navigation shortcut: Users can "enter" a portal to change the main view to be the one displayed inside the portal. Aside from serving as visual hyperlinks, portals can also be used as bookmarks (also called indexes) [BHP$^+$96] as well as a variety of nested multiple-view constructs including filters [BHP$^+$96, SFB94], magnifying glasses [Bel00, SFB94], overview and detail views [CMS98], and coordinated views [Bel00, SFB94]. We refer to nested constructs that can be created using portals in multiscale environments collectively as *portal tools*.

[1] Many other names have been used in the literature to refer to similar concepts, including "Magic Lenses" [BSP$^+$93, SFB94] and "wormholes" in Tioga-2 [ACSW96], the predecessor to DataSplash.

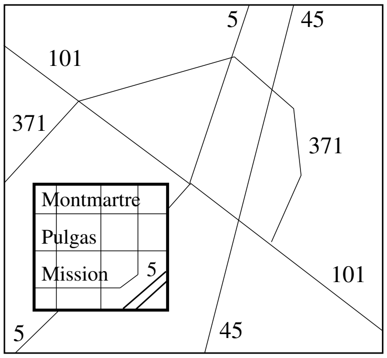

Figure 1: Magnifying glass tool.

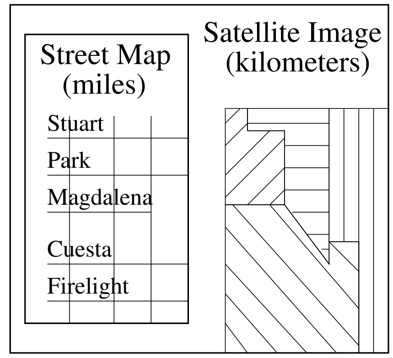

Figure 2: Coordinated view tool.

1.1 Portal Tools and Complexity

Portal tools often obey an underlying spatial model that ties together the views shown at different levels of nesting and dictates how the views change in response to user actions. For example, consider a typical magnifying glass tool, which covers a region of the world displayed on the screen and in its place displays an increased-scale representation of the occluded region. Figure 1 illustrates a cartographic application with a magnifying glass portal tool. Users may reposition the magnifying glass, causing its contents to shift, and the relationship between the change in position on the screen and the shift in contents of the magnifying glass portal is based on an implicit spatial model that incorporates scale as well as horizontal and vertical position. Users may also change the scale of the world being explored, automatically triggering a corresponding change in the scale of the magnifying glass contents to maintain a fixed scale ratio derived from a spatial model.

Different types of portal tools exhibit different spatial properties, *e.g.*, bookmark tools typically behave differently than magnifying glasses when repositioned by a user. Moreover, the behaviors of some tools can be complex. For example, consider coordinated view tools, which contain an inner world consisting of an alternative representation of the portion of the outer information world currently displayed on the screen. Figure 2 shows an example of a coordinated view tool that might be used in a GIS application to show a street map (inner world) in conjunction with a satellite image (outer world) of the same area. In many applications users may *pan*, or scroll, in the information world, causing the view of the outer world to shift while the coordinated view tool remains stationary on the screen. Meanwhile, the content of the coordinated view is shifted automatically to maintain the inner and outer views coordinated. Since the coordinated view tool remains fixed on the screen, when a pan occurs the position of the tool in the coordinates of the outer information world changes. However, if a user repositions the coordinated view tool directly, also altering its position relative to the outer information world, its contents remain unchanged. This apparent inconsistency in behavior has the useful property of allowing the tool to be freely repositioned on the screen to avoid occluding essential information without affecting the contents of the coordinated view tool, which continues to display an alternative representation of the portion of the outer information world currently displayed on the screen. In some applications, complexities such as this are warranted due to useful properties that arise such as the ability to affect which information is occluded without resorting to navigation. In fact, behaviors that are complex to describe may seem quite intuitive to expert users. On the other hand, for simple applications or novice users, it may be appropriate to remove complexities by, for example, locking the screen position of a coordinated view tool, which requires altering the way the tool behaves in response to user actions.

Even a single type of portal tool such as a coordinated view can be alternatively configured to behave in one of several ways. Interestingly, some configurations may result in asymmetric properties. For example, consider a magnifying glass tool, illustrated in Figure 1, in which the magnified scale of the nested world inside the magnifying glass is typically a function of the current scale of the outer world. Therefore, when a user alters the scale of the outer world, the scale of the inner world changes automatically, and the relationship is typically governed by a multiplicative magnification factor. In some instances, users are permitted to alter the scale of the inner world directly. If so, the portal may be configured so that the change, divided by the magnification factor, propagates to the outer world, having the same overall effect that would have occurred had the user altered the scale of the outer world directly and leaving the magnification factor unchanged. On the other hand, in an alternative configuration, a change made by a user to the scale of the inner world does not propagate to the outer world, having the effect of programming the magnification factor of the magnifying glass. In this alternative configuration, changing the scales of the inner and outer worlds are not symmetric actions: the former programs the magnification factor between the two worlds, while the latter alters the scales of both worlds while holding the magnification factor constant.

1.2 Formalism to Combat Complexity

As we have seen in Section 1.1, different portal tools exhibit different spatial behaviors in response to user actions, a single type of tool can be configured to exhibit several alternative behaviors, and these behaviors can be complex and asymmetric. Due to the presence of significant complexity, we feel that an overarching formal model for portal behavior that describes the space of possible behaviors in a systematic way would be of significant benefit to researchers, designers, programmers, and users. First, researchers of nested multiscale interfaces can leverage a formal model to help them understand, describe, communicate, and reason about properties of portal tools and the alternative configurations available. Second, by providing a clear enumeration and description of possible behaviors, a good formalism can assist designers faced with the nontrivial tasks of deciding which behaviors to make available, how best to present the available options to end-users, and what control mechanisms to provide, when offering portal tools in a nested multiscale interface environment. A formal model may also be of significant help to programmers of multiscale interfaces, which can use it as a basis for a simple and elegant implementation of portals with flexible configurations. Finally, a spatial model of portal behavior can be incorporated into documentation to guide non-expert users of systems like DataSplash [WOA+01] as they create portal tools that have flexible, visually programmable behaviors.

This paper presents a formal model for the spatial properties of nested user interface constructs. Although nesting has been proposed in a variety of contexts, we are aware of no previous attempt to model spatial relationships among nested information worlds formally. Our initial work in [OW00] developed a basic model for nested interfaces based on Boolean properties that describe whether certain actions such as navigating in worlds or repositioning portals trigger other actions to occur. The abstract model of [OW00] is intended to be very general, and does not specify the exactly way in which actions are correlated, which depends on the specific spatial or non-spatial model in use. In this paper we describe a detailed spatial model based on space-scale diagrams [FB95] that captures the semantics of a wide variety of portal tools and configurations in a unified way. The specific contributions of this paper are:

- We extend space-scale diagrams, originally designed for analyzing non-nested multiscale interfaces, to incorporate nesting in a natural way (Section 2).
- Drawing from our experience with a nested multiscale data visualization environment, we propose a model describing the way portals behave in response to user actions based on geometric relationships among elements in a space-scale diagram (Section 3).
- We provide a comprehensive taxonomy of 32 varieties of portal tools that obey our spatial model, and

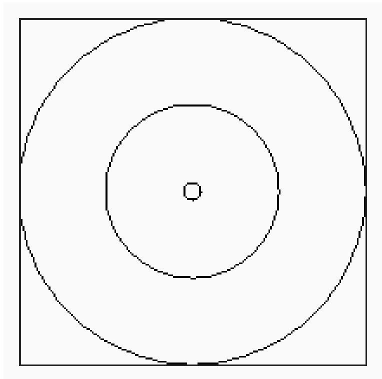

Figure 3: Example canvas.

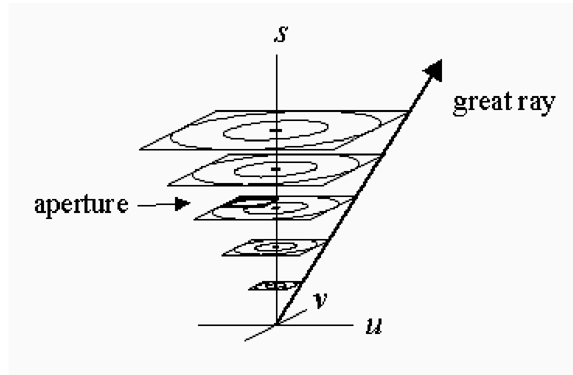

Figure 4: Space-scale diagram.

provide an accompanying online Java demonstration program for readers to experiment with different tool types and configurations in our taxonomy (Section 4).

- We propose techniques for animating the entrance into a portal while maintaining the user's orientation that exploit our spatial model (Section 5).

We now proceed with a short review of non-nested multiscale interfaces and space-scale diagrams, and then introduce nesting formally.

2 Multiscale Interfaces, Space-Scale Diagrams, and Nesting

In a typical multiscale interface a user navigates within a two-dimensional world that can be very large or even infinite. For our purposes it is convenient to think of it as a finite, two-dimensional rectangle with x and y coordinates called a *canvas*[2]. Figure 3 shows a simple canvas containing three concentric circles. The user is able to view portions of the canvas inside a fixed-size rectangle of screen real-estate called the *viewer window*. To control which portion of the two-dimensional canvas appears in the viewer window, the user can navigate in three dimensions by *panning* and *zooming*. The effect of panning is to change the portion of the canvas that appears in the viewer window without affecting the degree of magnification, or *scale*. The effect of zooming is to change the scale, causing a greater or smaller portion of the canvas to become visible in the viewer window.

2.1 Review of Space-Scale Diagrams

Multiscale interfaces are perhaps easiest understood with the help of *space-scale diagrams*, introduced by Furnas and Bederson [FB95], which represent scale explicitly, leading to a convenient way to model navigation and, in particular, zooming. We review space-scale diagrams briefly here (the reader is referred to [FB95] for a more comprehensive description and discussion). Figure 4 shows a space-scale diagram for the canvas in Figure 3. The two-dimensional canvas appears multiple times in the three-dimensional diagram that has dimensions u, v, and s, and each copy of the canvas lies in a different u–v plane and is scaled by a different amount. Scale is determined by the s coordinate. In the u–v plane at scale coordinate $s = 1$, the canvas is drawn at its "natural" scale, so that $u = x$ and $v = y$. At other s positions, the x and y dimensions are scaled multiplicatively (linearly) by s: $u = x \cdot s$ and $v = y \cdot s$. Due to this scaling effect, an (x, y) point

[2] Also called a "surface" in Pad [PF93].

in a canvas becomes a *great ray* (see Figure 4) in (u, v, s)-space originating at $(0, 0, 0)$ and passing through all points on the line parameterized by s as follows: $u = x \cdot s$; $v = y \cdot s$. Objects in the canvas such as the circles in Figure 3 become generalized cones in the space-scale diagram.

The portion of the canvas currently displayed on the screen in the viewer window is represented in a space-scale diagram as a 2D shape in u and v of fixed size and shape called an *aperture*, which is usually rectangular, as shown in Figure 4. The canvas is rendered by taking the portion that lies inside an aperture called the *parent frame* (for reasons that will soon become clear) and drawing it on the screen, scaled to fit exactly inside the viewer window. (The shape of the parent frame aperture must match the shape of the viewer window.) Since apertures are fixed in size and shape, their coordinates can be encoded by their center position, a (u, v, s) point that can be edited by the user via navigation (panning and zooming) operations. Editing the position of the parent frame aperture causes the contents of the viewer window to change.

2.2 Nesting in Space-Scale Diagrams

We can extend space-scale diagrams in a natural way to represent nested canvases (those containing portals). For simplicity, we assume only a single level of nesting, whereby a single *child canvas* is nested within a single *parent canvas*. Our model can be extended to incorporate multiple levels of nesting, as found in some environments such as [BHP$^+$96, WOA$^+$01]. We also simplify our discussion by focusing on parent canvases containing a single portal, although in general canvases may contain multiple portals, possibly linking to different child canvases.

Portals can be thought of as having two aspects: outline and contents. An outline in the parent canvas designating where the contents of a portal are to be drawn is called the *portal frame*. As with all objects in the parent canvas, a portal frame has an (x, y) location, a shape, and a size, all of which can potentially be edited by the user. The contents of a portal are defined by the *child frame*, an aperture (fixed-size 2D shape) in the space-scale diagram of the child canvas. Figure 5(a) shows an example of a space-scale diagram containing the three frames (parent, portal, child) of our model (the canvas contents have been omitted from the space-scale diagram to reduce clutter), augmented with a *recursive* portal. A recursive portal is one in which the child canvas is the same as the parent canvas, so the portal goes to another location and/or scale in the same canvas. With recursive portals, all three frames to be drawn in the same space-scale diagram. (A non-recursive portal can be drawn using two diagrams to represent the parent and child canvases.) The reason the portal frame has (x, y) coordinates while the other two frames have (u, v, s) coordinates is that the portal frame, like any graphical object within a canvas, is manifest at all scales while the parent and child frames each have a corresponding scale. For simplicity, we assume throughout this paper that all three frames in our model are rectangular, as is common, so the portal frame appears in the space-scale diagram as a pyramid composed of four great rays.

Figure 5(b) shows the viewer window as seen by the user when the parent and child canvases are both the one shown in Figure 3 and are rendered using the frames shown in Figure 5(a). A portal is rendered for the user by taking the region of the child canvas enclosed by the child frame, scaling it appropriately, and drawing it inside the portal frame in the parent canvas, which in turn is rendered on the screen inside the viewer window as discussed in Section 2.1. (The shape of the child frame must match the shape of the portal frame.) In Figure 5(a), the cross-section of the portal frame taken at the current scale coordinate of the parent frame is smaller than the parent frame, and is enclosed by the parent frame. Consequently, the portal outline is rendered inside the viewer window in Figure 5(b). The cross-section of the portal frame at the scale of the parent frame is also smaller than the child frame, meaning that the portion of the canvas bounded by the child frame is demagnified (reduced in scale) when rendered inside the portal in Figure 5(b). To enable rendering and simplify animation of portal entering (see Section 5), the aspect ratios of the three rectangular frames should match, and for simplicity we assume that all three aspect ratios remain fixed and equal at all times.

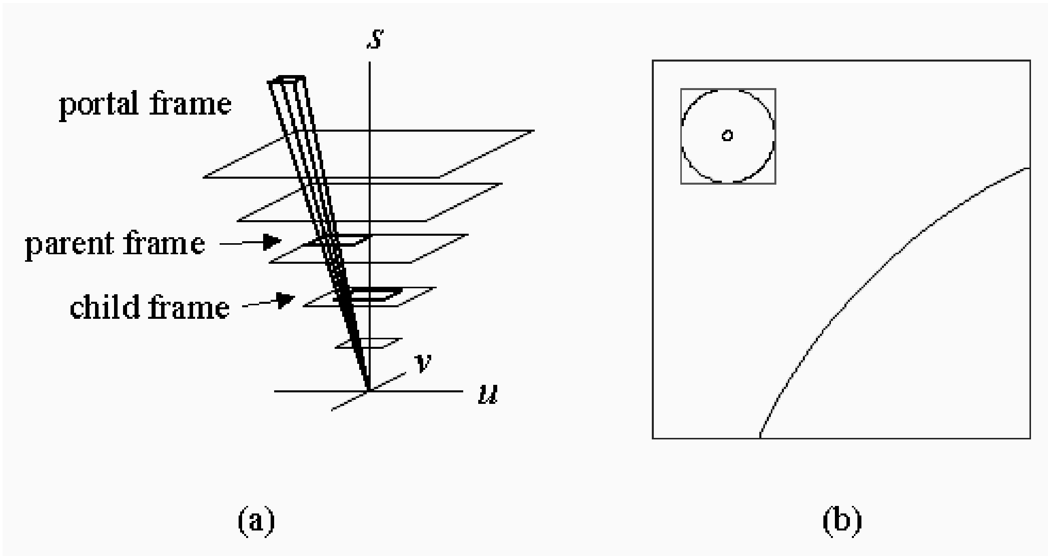

Figure 5: A space-scale diagram containing the three frames of our model for a canvas with a recursive portal (a), and the corresponding viewer window (b).

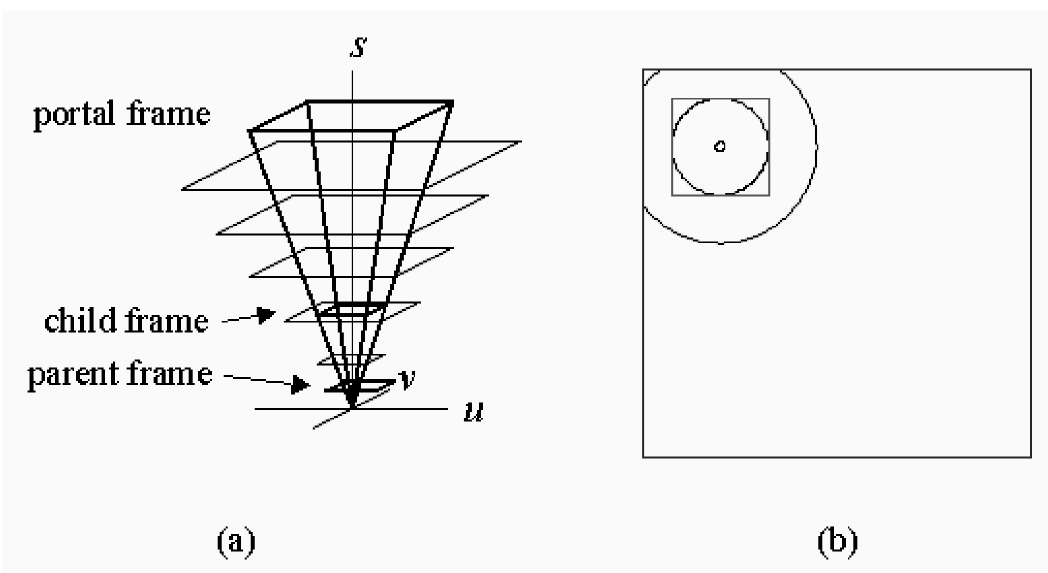

Figure 6: A space-scale diagram showing a new interface state (a), and the corresponding viewer window (b), obtained by beginning with Figure 5 and editing either the parent or portal frame with the parent-portal latch active.

Our basic three-frame model for a parent canvas containing a single portal to a child canvas was originally introduced in [OW00], which presented an equivalent notion of frames not based on space-scale diagrams. As justified in [OW00], the three frames can be assigned the ordering *parent frame* $\succ$ *portal frame* $\succ$ *child frame* based on the conceptual distance from the user: the parent frame encloses the main display, the portal frame is an element inside that display, and the portal frame is conceptually located beneath the main display, inside the portal frame. For convenience, we use the notation $X+$ (and $X-$) to refer to the frame immediately succeeding (preceding) frame X in the conceptual order. Next we discuss user interaction.

3 Spatial Model for Behavior During User Interaction

One way in which users can interact with nested multiscale interfaces is by editing the coordinates of the three frames in our model. Editing the (u, v, s) position of the parent frame, which always remains fixed in size, is accomplished via *parent navigation* operations, *i.e.*, pan and zoom. Many environments including [BHP+96, WOA+01] also support *child navigation*, which allows users to edit the (u, v, s) position of the child frame. For example, a user could potentially pan and zoom the child canvas in Figure 5(b) to enlarge and recenter its contents without affecting the parent canvas. Panning either the parent or child frames alters the u and v position in the space-scale diagram of the center of the frame in a manner independent of the current scale. For example, a "pan left" operation performed at any scale will set $u := u - K$ for some constant K that does not depend on s. Zooming the parent or child frame changes the s position of the frame, typically while keeping it centered on the same great ray, thereby holding the x and y coordinates constant. Therefore, whenever zooming is performed, changing the scale from s to s', the other two coordinates are updated automatically to $u' = u \cdot \frac{s'}{s}$ and $v' = v \cdot \frac{s'}{s}$.

In addition to editing the parent and child frames via parent and child navigation, users can edit the portal frame by manipulating the portal object. A portal can be thought of as an object having the special property that it displays another canvas, and some environments including [BHP+96, WOA+01] permit users to move and resize portal objects in the same manner as other non-portal objects. For example, a user could make the portal in Figure 5(b) larger and reposition it. The effect on the space-scale diagram would be to enlarge and reposition the pyramid representing the portal frame. Under the simplest of behaviors, each frame can be freely edited by the user and each is independent, meaning that edits to one frame have no effect on the other two frames. However, many other behaviors are possible in which the user is prevented from editing certain frames, or, more interestingly, in which editing one frame causes changes to other frames. We now describe two types of fundamental properties, frame editability and dependencies, that govern these more complex behaviors.

3.1 Frame Editability and Dependencies

As introduced in [OW00], there are two types of Boolean properties governing the fundamental behavior of portals during user interaction. Each frame has an editability property, and each ordered pair of frames has a dependency property, resulting in nine Boolean properties overall. We review these basic properties here before introducing our detailed model for frame dependencies based on spatial relationships among frames in Section 3.2.

The user can be prevented from editing a frame by disabling the frame's *editability property*. The editability of each frame can be configured independently. By default, editing one frame has no effect on the other two frames, although it may change the contents of the viewer window. For example, making the portal in Figure 5(b) smaller does not affect the child frame, which remains in the same position in the space-scale diagram, so the same contents (two concentric circles) would be displayed in the smaller

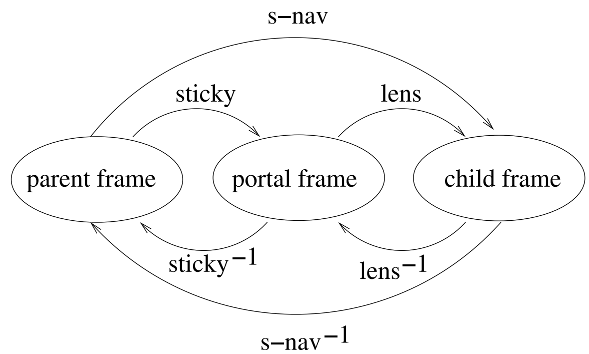

Figure 7: The three frames in our model, in order of conceptual distance from the user, with arcs between them representing potential dependencies between frames.

portal frame. *Frame dependencies* are properties that cause one frame to change automatically when the user edits another frame. Dependencies are one-way links between ordered pairs of frames, of which there are six. Each dependency can either be enabled or disabled. If a dependency from frame A to frame B, written $A \rightarrow B$, is enabled, then whenever the user edits the position and/or size of frame A, the position and/or size of frame B changes automatically. Of course, a dependency $A \rightarrow B$ cannot be enabled unless frame A is editable. On the other hand, even if frame B is not editable, the dependency can still be enabled. Editability only restricts direct editing, while still allowing indirect editing via frame dependencies.

There are six possible frame dependencies in our three-frame model. We classify the dependencies into forward and reverse dependencies. A dependency $A \rightarrow B$ is forward if $A \succ B$, and reverse otherwise. Each forward dependency $A \rightarrow B$ has an inverse, $B \rightarrow A$ that is a reverse dependency. It is helpful to think of the frame dependency properties of a portal as a directed graph with three vertices, one for each frame (parent, portal, and child), as shown in Figure 7 with frames displayed in order of conceptual distance from the user, from left to right. Each arc in the graph is labeled with the name we give to the corresponding dependency. The three forward dependencies are named as follows: *parent* $\rightarrow$ *portal* $\equiv$ *sticky*, *parent* $\rightarrow$ *child* $\equiv$ *s-nav* (for synchronous navigation), and *portal* $\rightarrow$ *child* $\equiv$ *lens*. We name their inverses *sticky*$^{-1}$, *s-nav*$^{-1}$, and *lens*$^{-1}$, respectively.

There are a total of 125 distinct fundamental behaviors, considering only the Boolean editability and dependency properties, and accounting for the restriction that outgoing dependencies are only allowed from editable frames. Many of these behaviors are unintuitive and can be confusing, and three *usability rules* were proposed in [OW00] to eliminate unintuitive behaviors, reducing the number of allowed behaviors to 32. These 32 intuitive behaviors are abstract, in the sense that our general model from [OW00] only considers whether dependencies between frames are enabled or disabled, and not what effect enabled dependencies have, which depends on the way dependencies are implemented.

The appropriate dependency implementation depends on the characteristics desired, which vary across applications. In some applications, the relationships among frames may not have any spatial significance. For example, consider a political map visualization of the United States in which a recursive portal displays the home town of the governor of the state displayed in the main viewer window. The portal exhibits, among others, the s-nav dependency because edits to the parent frame (navigation in the main viewer window) can trigger a change to the child frame (portal contents), but the relationship is not spatial. On the other hand, in many common and natural applications of nested multiscale interfaces, dependent frames exhibit a spatial relationship. We describe our model for a spatial implementation of frame dependencies next.

3.2 Spatial Implementation of Dependencies

Frame dependencies can be implemented in a way that obeys a spatial metaphor. There are at least two plausible metaphors for nested multiscale interfaces. In one, the portal frame is treated as a physical window, where moving closer spreads out the viewing angle, thereby enlarging the visible area of the child canvas. Alternatively, portals can be treated like hanging pictures rather than windows in this respect. Our original intuition when we helped develop portals in the DataSplash system [WOA$^+$01] was to follow the physical window metaphor due to its correspondence with windows in the real world. However, after extensive experimentation it became clear that the physical window model was impractical because the portal contents could not be made invariant during zooming, making it difficult to program useful nested visualizations. Therefore, it was decided that the hanging picture metaphor would likely be superior for most purposes. We now present our spatial model for implementing frame dependencies, which is based on the hanging picture metaphor, and which satisfies the mapping transitivity and inversion properties defined in [OW00] and deemed essential for intuitive nested interface environments. For simplicity, we focus on recursive portals, but our model extends easily to non-recursive ones.

3.2.1 Latches

The basic conceptual constructs in our spatial model are *latches*, which temporarily fasten together pairs of frames. There are two latches in our three-frame model, one between each pair of frames that are adjacent in conceptual order: the *parent-portal latch* and the *child-portal latch*. Each of the two latches involves one aperture (parent or child frame), represented as a fixed-size rectangle in the space-scale diagram, and the portal frame, represented as a pyramid of variable size. Consider the four points at which the four great rays making up the pyramid of the portal frame intersect the plane of one of the aperture frames. These are called the *intersection points* of the aperture frame, and their four (u, v) coordinates relative to the (u, v) position of the aperture frame center are called the *relative intersection coordinates*. When a latch is active, it fastens the relevant aperture frame to the portal frame by maintaining the relative intersection coordinates invariant, causing the movements of the two frames to be coordinated.

By physical analogy, an aperture frame can be thought of as being embedded in a thin, flat plate pierced by four rods representing the great rays of the portal frame, the ends of which are anchored to the origin. The holes in the plate made by the rods represent the relative intersection coordinates, which remain in the same position in the plate. The rods are permitted to slide freely up and down in the holes. When the latch is active, the effect is that moving the plate laterally (panning the aperture) causes the rods (portal frame) to move along with it. Moving the plate up (increasing the scale coordinate of the aperture by zooming) causes the angle between the rods to decrease and the portal frame to become smaller. Conversely, moving the plate down (decreasing the scale coordinate of the aperture by zooming) spreads the rods farther apart and causes the portal frame to grow larger, as illustrated in the transition from Figure 5 to Figure 6, in which both panning and zooming of the parent frame aperture have been performed with the parent-portal latch active. Edits made to the portal frame transfer to the aperture frame via the same mechanism when the latch is active. Moving the rods in unison laterally (repositioning the portal frame) causes the plate to move along with them, indirectly panning the aperture. Bringing the rods closer together (reducing the size of the portal frame) causes the plate to move up, increasing the scale coordinate of the aperture such that the relative intersection coordinates remain unchanged. Conversely, spreading the rods apart (increasing the size of the portal frame) causes the plate to move down, decreasing the scale coordinate of the aperture. This effect is illustrated in the transition from Figure 5 to Figure 6, which could have been accomplished by spreading out the portal frame rods with the parent-portal latch active.

We now describe the effect of latches mathematically. Let the parent aperture frame coordinates be (u_p, v_p, s_p), and the coordinates of the lower-left and upper-right corners of the portal frame be (x_1, y_1)

Dependency		*Latch Activated*
sticky	(*parent* $\rightarrow$ *portal*)	parent-portal
s-nav	(*parent* $\rightarrow$ *child*)	child-portal
lens	(*portal* $\rightarrow$ *child*)	child-portal
sticky^{-1}	(*portal* $\rightarrow$ *parent*)	parent-portal
s-nav^{-1}	(*child* $\rightarrow$ *parent*)	parent-portal
lens^{-1}	(*child* $\rightarrow$ *portal*)	child-portal

Table 1: List of which latch is activated for each enabled dependency $A \rightarrow B$ during edits to frame A.

and (x_2, y_2), respectively. The four relative intersection coordinates are, in clockwise order beginning with the lower-left: $(x_1 \cdot s_p - u_p, y_1 \cdot s_p - v_p)$, $(x_1 \cdot s_p - u_p, y_2 \cdot s_p - v_p)$, $(x_2 \cdot s_p - u_p, y_2 \cdot s_p - v_p)$, and $(x_2 \cdot s_p - u_p, y_1 \cdot s_p - v_p)$. If the user edits the parent frame, changing its coordinates from (u_p, v_p, s_p) to (u'_p, v'_p, s'_p), and the parent-portal latch is active, then the lower-left and upper-right (x, y) portal frame coordinates change to $\left(\frac{x_1 \cdot s_p + (u'_p - u_p)}{s'_p}, \frac{y_1 \cdot s_p + (v'_p - v_p)}{s'_p}\right)$ and $\left(\frac{x_2 \cdot s_p + (u'_p - u_p)}{s'_p}, \frac{y_2 \cdot s_p + (v'_p - v_p)}{s'_p}\right)$, respectively. It is easy to verify that these new portal frame coordinates leave the relative intersection coordinates between the portal and parent frames unchanged. If the user edits the portal frame, changing its lower-left and upper-right coordinates from (x_1, y_1) and (x_2, y_2) to (x'_1, y'_1) and (x'_2, y'_2), respectively, and the parent-portal latch is active, then the parent frame coordinates change to:

$$\left(u_p + s_p \cdot \frac{x_2 \cdot x'_1 - x_1 \cdot x'_2}{x'_2 - x'_1}, v_p + s_p \cdot \frac{y_2 \cdot y'_1 - y_1 \cdot y'_2}{y'_2 - y'_1}, s_p \cdot \frac{x_2 - x_1}{x'_2 - x'_1}\right)^3$$

As before, it can be verified that the relative intersection coordinates between the portal and parent frames remain unchanged. The corresponding formulae for the child-portal latch are similar.

As we will see in Section 5, the concept of latches is useful for creating an animation sequence for entering a portal. The primary purpose of latches, however, is to link together the motions of dependent frames during editing by the user, as discussed next.

3.2.2 Activation of Latches

Latches are inactive by default, and they may be activated temporarily during frame editing, depending on which dependencies are enabled. While the user is editing frame A, a latch is activated for each outgoing dependency $A \rightarrow X$ that is enabled. If $A \rightarrow X$ is a forward dependency, *i.e.*, $A \succ X$, then the latch between X and $X-$ (the frame immediately preceding X in the conceptual order) is activated while frame A is edited. Otherwise, if $A \rightarrow X$ is a reverse dependency, *i.e.*, $X \succ A$, then the latch between X and $X+$ (the frame immediately succeeding X in the conceptual order) is activated. Put simply, if any dependency $A \rightarrow X$ is enabled, then while frame A is edited, the latch between X and the frame before X on the path from A to X is activated. For example, suppose the parent frame is edited, *i.e.*, the user pans or zooms in the parent canvas. If the sticky dependency (*parent* $\rightarrow$ *portal*) is enabled, then the parent-portal latch is activated during editing. If the s-nav dependency (*parent* $\rightarrow$ *child*) is enabled, then the child-portal latch is activated. If both dependencies are enabled, then both latches are activated while the parent frame is edited. Table 1 lists, for each dependency $A \rightarrow B$, which latch is activated while the user edits frame A. Note that latch activation is not necessarily symmetric, *e.g.*, if the parent-portal latch is activated while the user edits the parent frame, it may not necessarily be activated while the user edits the portal frame.

[3] Note that the new scale coordinate can be defined equivalently in terms of either x or y since we require the aspect ratio of the parent frame to remain fixed.

Forward Dependencies Enabled	*Recursive Portal Tool Type*	*Non-recursive Portal Tool Type*
{}	canvas-stationary visual hyperlink	
{sticky}	viewer-stationary visual hyperlink (bookmark)	
{lens}	canvas-stationary magnifying glass	canvas-stationary filter/transformer
{sticky, s-nav, lens}	viewer-stationary magnifying glass	viewer-stationary filter/transformer
{sticky, s-nav}	overview	coordinated view

Table 2: The five allowed forward dependency combinations, and some types of recursive and non-recursive portal tools that use them.

The two latches are spatial constructs that implement the six dependencies from our abstract model from [OW00] to define a new spatial model. It may seem surprising that no latches are needed between the parent and child frames in our spatial model. According to one of the fundamental usability rules established in [OW00], dependencies $A \rightarrow B$ that bypass intermediate frames in the conceptual order must be accompanied by dependencies from A to each intermediate frame between A and B in the conceptual order. Therefore, when frame A is edited, all latches along the path from A to B will be activated, thereby transferring edits indirectly from A to B via the intermediate frames. For example, the s-nav dependency (*parent* $\rightarrow$ *child*) bypasses the portal frame, so the sticky dependency (*parent* $\rightarrow$ *portal*) must also be enabled whenever the s-nav dependency is enabled. The combination of those two dependencies causes both the parent-portal and child-portal latches to be activated while the parent frame is edited, thereby causing the edits to transfer indirectly from the parent frame to the child frame via the portal frame. In our physical analogy, the parent and child frames are each embedded in a thin plate, which are both pierced by the same four rods representing the great rays of the portal frame. When both latches are active, moving one plate moves the rods, whose motions indirectly cause the other plate to move.

Perhaps the best way to understand our spatial model based on latches, which dictates how portals behave during user interaction when dependencies are enabled, is to interact with our online demonstration program, available at [Ols02]. Users of our demonstration program can configure the editability and dependency properties, edit frames, and observe how edits transfer indirectly to other frames via latches. Two visual representations are provided of the nested canvas being controlled: a space-scale diagram containing the three frames in our model and the viewer window as rendered for the end user. Figures 5 and 6 are screenshots of our program.

4 Taxonomy of Portal Tools Employing Our Spatial Model

In this section we provide a taxonomy of portal tools that can be created by applying our spatial dependency implementation based on latches to one of the 32 fundamental behaviors that satisfy the usability rules of [OW00]. We classify portal tools along two main dimensions: type of tool and programmability.

4.1 Tool Type

The type of tool is defined by which forward dependencies are enabled. There are three forward dependencies, and therefore eight combinations, but only five of them are permitted by the usability rules of [OW00]. Table 2 lists the five legal combinations of forward dependencies, along with some types of tools that can be created with a recursive or non-recursive portal that has exactly that set of forward dependencies enabled and obeys our spatial dependency implementation.

We now briefly describe the different types of portal tools listed in Table 2 (the reader is referred to [OW00] for some discussion of applications that might use the different types of tools).

- *Visual hyperlinks* are analogous to hypertext hyperlinks (*e.g.*, between Web pages) in that they allow the user to navigate instantly from a location on one canvas to a new location on the same canvas, or some location on another canvas. Additionally, visual hyperlinks display the contents of the destination in a sub-window, providing a "preview" of the destination. Hyperlinks use no forward dependencies.

- *Filter/transformers* [BHP$^+$96, SFB94] are portals that show an alternative representation of the region of the parent canvas that is occluded by the portal. For this type of tool, the lens dependency is enabled, causing the contents of a filter/transformer to be correlated with its position and size in the parent canvas.

- *Magnifying glasses* [ACSW96, PF93] show a magnified view of the region of the canvas underneath. See Figure 1 for an illustration. As with filter/transformers, the lens dependency is enabled, causing the contents of a magnifying glass to be correlated with its position and size in the outer canvas.

- *Overview tools* are fixed on the screen and show a demagnified view of the portion of the canvas surrounding the current viewer window contents, creating an overview and detail view [CMS98]. The s-nav dependency is enabled, causing the contents of an overview to be correlated with the contents of the main viewer window. The sticky dependency is also enabled, causing an overview tool to remain the same size and in the same position on the screen despite parent navigation.

- *Coordinated views* [Bel00, SFB94] remain on the screen at all times and show an alternative representation of the region of the parent canvas displayed in the main viewer window. See Figure 2 for an illustration. In contrast to filter/transformers, the view displayed inside a coordinated view tool depends only on what is displayed in the main viewer window, and is not related to the position of the portal. As with overview tools, coordinated views have the sticky and s-nav dependencies enabled, causing them to remain stationary on the screen and their contents to be correlated with the contents of the viewer window.

Visual hyperlinks, filter/transformers, and magnifying glasses can be either *canvas-stationary* or *viewer-stationary*. (Overview tools and coordinated views are always viewer-stationary by nature.) The position and size of a canvas-stationary tool are fixed relative to the canvas, and therefore the observed position relative to the viewer window changes during panning in the parent canvas and the observed size changes during zooming. In contrast, the position and size of viewer-stationary tools relative to the viewer window remains fixed while parent navigation is performed. Canvas-stationary portal tools can be made viewer-stationary by enabling the sticky dependency and taking the transitive closure of the resulting dependency graph to guarantee dependency transitivity as required by the usability rules of [OW00]. Viewer-stationary visual hyperlinks are also called *bookmarks* (known as *indexes* in Pad++ [BHP$^+$96]). For simplicity, throughout the remainder of this paper we focus on recursive portal tools and commonly refer to the five tool types by abbreviation: hyperlinks, bookmarks, canvas magnifiers, viewer magnifiers, and overviews.

4.2 Programmability Options

Before we describe the second component of our taxonomy, frame programmability, we must first introduce the concept of interface state, upon which our notion of programmability is based. Let (x_1, y_1) and (x_2, y_2) be the lower-right and upper-left corners, respectively, of the portal frame, and let (u_p, v_p, s_p) and (u_c, v_c, s_c) be the coordinates of the parent and child frames, respectively. The *interface state* $\mathcal{S}$ of a parent canvas containing a nested child canvas specifies the current coordinates of all three frames in our model. However, it is useful to reformulate the interface state in terms of relative relationships among frames. Recall from Section 3.2.1 that the points at which the portal frame intersects the plane of an aperture (parent or child frame), in (u, v) coordinates relative to the aperture center are called the relative intersection coordinates. Since we require portals to have a fixed aspect ratio, we can encode the parent-portal relative intersection coordinates as the three-tuple $I_p = \langle x_1 \cdot s_p - u_p, y_1 \cdot s_p - v_p, x_2 \cdot s_p - u_p \rangle$, and the child-portal relative intersection coordinates similarly as $I_c = \langle x_1 \cdot s_c - u_c, y_1 \cdot s_c - v_c, x_2 \cdot s_c - u_c \rangle$. The overall interface state can be rewritten by encoding the coordinates of each frame relative to those of the previous frame in the conceptual order. The result is the following three-tuple of three-tuples: $\mathcal{S} = \langle \mathcal{S}_{parent}, \mathcal{S}_{portal}, \mathcal{S}_{child} \rangle = \langle \langle u_p, v_p, s_p \rangle, I_p, I_c \rangle$. The three components of the interface state, $\mathcal{S}_{parent}$, $\mathcal{S}_{portal}$, and $\mathcal{S}_{child}$, each correspond to one frame in our model, and together they specify unambiguously the coordinates of all three frames. $\mathcal{S}_{parent}$ specifies which part of the parent canvas is displayed inside the viewer window and $\mathcal{S}_{portal}$ specifies the placement and size of the portal within the viewer window. $\mathcal{S}_{child}$ pertains to the contents of the portal and specifies which portion of the child canvas is displayed inside the portal, relative to the placement and size of the portal outline within the parent canvas. For example, with a magnifying glass, $\mathcal{S}_{child}$ determines the magnification factor as well as the offset between the portion of the outer canvas occluded by the magnifying glass and the portion of the canvas displayed inside it.

When a forward dependency between two frames is enabled, the interface state component of the dependent frame encodes the spatial relationship between the two frames. For example, in the case of the lens dependency for magnifying glasses, $\mathcal{S}_{child}$ encodes the magnification factor and offset. Altering the interface state of the dependent frame can be thought of as a way to program the spatial relationship between the frames. When a frame is not dependent on any other frames, spatial relationships among frames are not relevant because its coordinates remain fixed unless it is edited directly, as with the child frame of a bookmark. In that case, altering its interface state can be thought of as a way to program its absolute coordinates. In either case, we refer to the act of altering a frame's interface state by editing that frame as *programming* the frame.

We say that frame A is *programmable* if users can alter $\mathcal{S}_A$ directly by editing A. If all frames are editable and no reverse dependencies are enabled, then all frames are programmable. Frame A can be made nonprogrammable in one of two ways. The first way is to make frame A non-editable, which has the indirect consequence of disabling any outgoing dependencies from A of the form $A \to X$ (recall from Section 3.1 that dependencies from non-editable frames are not allowed). Second, if any incoming forward dependencies to frame A are enabled, then A can be made nonprogrammable by *adding inversion*, which proceeds in two steps. First, for each forward incoming dependency $X \to A$ enabled, we enable its inverse $A \to X$, which is a reverse dependency. Second, to guarantee dependency transitivity as required by the usability rules of [OW00], we take the transitive closure of the resulting dependency graph. Adding inversion is only applicable if at least one forward dependency into frame A is enabled. Note that both methods for making a frame nonprogrammable, adding inversion and disabling editability, can result in a change to the tool type by affecting which forward dependencies are enabled. Adding inversion can in some cases lead to additional forward dependencies being enabled due to the transitive closure step. Disabling editability can lead to one or more forward dependencies being disabled since outgoing dependencies from non-editable frames are not permitted. We revisit this apparent oddity below in Section 4.3.

To illustrate the two options for making frames nonprogrammable, we take as an example a fully programmable canvas magnifier, which has only the lens dependency enabled. Since a magnifying glass shows a detailed view of a small area of the canvas, users may wish to explore the immediate neighborhood of that view without affecting the outer view by panning inside the magnifying glass. Performing this action amounts to editing the child frame, which alters $\mathcal{S}_{child}$ and programs the offset between the position of the magnifying glass in the canvas and the location viewed within the magnifying glass. Normally, the offset of a magnifying glass is kept at zero, so programming the offset may, for novice users, result in unexpected behavior during further interaction. Therefore, it may be appropriate to disable programming of the child frame for novice users. For magnifying glasses, which have the lens forward dependency enabled, there are two ways to disable child frame programmability. The simplest is to disable editing of the child frame, disallowing users from panning or zooming the contents of the portal directly. The second way to make the child frame nonprogrammable is to add inversion by enabling the lens^{-1} dependency. In the resulting behavior, if a user pans the contents of the magnifying glass, the magnifying glass itself moves within the outer canvas to remain centered over the portion of the canvas shown in the magnified inner view. The effect is the same as if the user had instead repositioned the magnifying glass over a new area of the outer canvas, keeping the offset fixed at zero.

Both options for making the child frame nonprogrammable are possible with canvas-stationary magnifying glasses, but for tools with no forward dependencies into the child frame enabled, it is not possible to add inversion. For example, consider a bookmark tool, which does not use the lens dependency. It is not appropriate to attempt to prevent the user from programming the bookmark contents (child frame) by enabling the lens^{-1} dependency, thereby causing the bookmark tool to be repositioned or resized in the parent canvas in response to child navigation. In fact, this behavior is not permitted by the usability rules.

In summary, there are five types of tools, defined by which forward dependencies are enabled. For each type of tool, each frame can be either programmable or nonprogrammable, depending on which frames are editable and which reverse dependencies are enabled. A frame can always be made nonprogrammable by disabling editability of that frame. In some cases, the same frame can alternatively be made nonprogrammable by adding inversion. Note that the programmability of the parent frame is a global option, which affects the entire interface by preventing the user from panning and zooming the parent canvas. In contrast, the programmability of the portal and child frames only pertains to an individual portal, and can be configured differently for each portal present in the interface. We now present our overall taxonomy of portal tools that obey our spatial model, which classifies them by tool type and programmability.

4.3 Portal Tool Taxonomy

Table 3 lists every allowed combination of tool type and programmability options, along with the resulting behavior. Behaviors are represented as three strings of three T/F values each. The first string encodes which forward dependencies are enabled, in the order sticky, s-nav, lens, and the second string encodes which reverse dependencies are enabled, in the order sticky^{-1}, s-nav^{-1}, lens^{-1}. The third string encodes which frames are editable, in the order parent, portal, child. For each tool type, the first eight rows correspond to the $2^3 = 8$ options available by making each frame either programmable (indicated by "Y"), or nonprogrammable by disabling editability (indicated by "N-noned"), which are always possible for each frame. Additional rows, if any, correspond to options resulting from making one or more frames nonprogrammable by adding inversion (indicated by "N-inv"), which is only possible in some cases. Each unique behavior obtained by selecting a tool type and choosing a programmability option for each frame is assigned a unique identification number. As discussed above, in some cases disabling programmability of frames (either by disabling editability or by adding inversion) may enable or disable one or more forward dependencies, causing the behavior to be equivalent to that of another tool type. We indicate such cases in Table 3 by referencing the identification number of the equivalent behavior in parentheses. While the number of valid tool type and programmability options is greater, there are 32 nonequivalent behaviors, accounting for all 32 behaviors that satisfy the usability rules of [OW00].

To illustrate how a portal tool can become equivalent to another due to making some aspect nonprogrammable, we present two examples. First, making the parent frame nonprogrammable by disabling editability removes the sticky and s-nav dependencies of any portal tools present since outgoing dependencies are not allowed in the absence of editability. Therefore, when the parent frame is not editable, a bookmark, which normally employs the sticky dependency, is equivalent to a hyperlink, which has the sticky dependency disabled (see the fifth through eighth rows for the bookmark tool type in Table 3). This equivalence makes intuitive sense: if the user is not permitted to pan or zoom the parent canvas, then there is no detectable difference between canvas-stationary and viewer-stationary tools.

Tool Type	Programmability Options Parent	Portal	Child	Behavior Forward Deps.	Reverse Deps.	Editability	Behavior ID
hyperlink	Y	Y	Y	FFF	FFF	TTT	1
	Y	Y	N-noned	FFF	FFF	TTF	2
	Y	N-noned	Y	FFF	FFF	TFT	3
	Y	N-noned	N-noned	FFF	FFF	TFF	4
	N-noned	Y	Y	FFF	FFF	FTT	5
	N-noned	Y	N-noned	FFF	FFF	FTF	6
	N-noned	N-noned	Y	FFF	FFF	FFT	7
	N-noned	N-noned	N-noned	FFF	FFF	FFF	8
bookmark	Y	Y	Y	TFF	FFF	TTT	9
	Y	Y	N-noned	TFF	FFF	TTF	10
	Y	N-noned	Y	TFF	FFF	TFT	11
	Y	N-noned	N-noned	TFF	FFF	TFF	12
	N-noned	Y	Y	FFF	FFF	FTT	(5)
	N-noned	Y	N-noned	FFF	FFF	FTF	(6)
	N-noned	N-noned	Y	FFF	FFF	FFT	(7)
	N-noned	N-noned	N-noned	FFF	FFF	FFF	(8)
	Y	N-inv	Y	TFF	TFF	TTT	13
	Y	N-inv	N-noned	TFF	TFF	TTF	14
canvas magnifier	Y	Y	Y	FFT	FFF	TTT	15
	Y	Y	N-noned	FFT	FFF	TTF	16
	Y	N-noned	Y	FFF	FFF	TFT	(3)
	Y	N-noned	N-noned	FFF	FFF	TFF	(4)
	N-noned	Y	Y	FFT	FFF	FTT	17
	N-noned	Y	N-noned	FFT	FFF	FTF	18
	N-noned	N-noned	Y	FFF	FFF	FFT	(7)
	N-noned	N-noned	N-noned	FFF	FFF	FFF	(8)
	Y	Y	N-inv	FFT	FFT	TTT	19
	N-noned	Y	N-inv	FFT	FFT	FTT	20
viewer magnifier	Y	Y	Y	TTT	FFF	TTT	21
	Y	Y	N-noned	TTT	FFF	TTF	22
	Y	N-noned	Y	TTF	FFF	TFT	(29)
	Y	N-noned	N-noned	TTF	FFF	TFF	(30)
	N-noned	Y	Y	FFT	FFF	FTT	(17)
	N-noned	Y	N-noned	FFT	FFF	FTF	(18)
	N-noned	N-noned	Y	FFF	FFF	FFT	(7)
	N-noned	N-noned	N-noned	FFF	FFF	FFF	(8)
	Y	Y	N-inv	TTT	FFT	TTT	23
	N-noned	Y	N-inv	FFT	FFT	FTT	(20)
	Y	N-noned	N-inv	TTF	FTT	TFT	(32)
	Y	N-inv	Y	TTT	TFF	TTT	24
	Y	N-inv	N-noned	TTT	TFF	TTF	25
	Y	N-inv	N-inv	TTT	TTT	TTT	26
overview	Y	Y	Y	TTF	FFF	TTT	27
	Y	Y	N-noned	TTF	FFF	TTF	28
	Y	N-noned	Y	TTF	FFF	TFT	29
	Y	N-noned	N-noned	TTF	FFF	TFF	30
	N-noned	Y	Y	FFF	FFF	FTT	(5)
	N-noned	Y	N-noned	FFF	FFF	FTF	(6)
	N-noned	N-noned	Y	FFF	FFF	FFT	(7)
	N-noned	N-noned	N-noned	FFF	FFF	FFF	(8)
	Y	Y	N-inv	TTF	FTT	TTT	31
	Y	N-noned	N-inv	TTF	FTT	TFT	32
	Y	N-inv	Y	TTT	TFF	TTT	(24)
	Y	N-inv	N-noned	TTT	TFF	TTF	(25)
	Y	N-inv	N-inv	TTT	TTT	TTT	(26)

Table 3: An exhaustive enumeration of every allowed combination of tool type and programmability options, along with the resulting behavior. Duplicate behaviors are indicated with parentheses around the behavior identification number. See the text for an explanation of the abbreviated notation.

As a second example, consider a coordinated view tool (as in Figure 2), which, like an overview tool, has both the sticky and s-nav forward dependencies enabled, and whose contents remain unchanged when the user edits the position or size of the portal in the viewer window. As suggested in Section 1.1, it may be appropriate to make the portal frame nonprogrammable to simplify the behavior and avoid confusing novice users. Making the portal frame nonprogrammable by adding inversion causes the sticky^{-1} dependency to be enabled, and taking the transitive closure causes the lens dependency to be enabled as well. Therefore, a coordinated view tool with a portal frame made nonprogrammable by adding inversion is equivalent to a viewer magnifier, which has all three forward dependencies enabled (see the last three rows of Table 3). This equivalence makes intuitive sense as well: the only difference between a coordinated view tool and a viewer magnifier is that with a viewer magnifier, the contents of the portal change when the user edits the portal's position or size relative to the viewer window. If the user is not permitted to reposition or resize the portal relative to the viewer window then the effective difference between the two tool types is removed. (Of course, although the two tool types share a common behavior in this case, they may be instantiated differently, *i.e.*, have opposite magnification factors.)

5 Animating the Entrance Into a Portal

An important feature in some nested multiscale interface environments is the ability of the user to "enter" a portal [WOA^{+}01], causing the parent canvas and frame to be replaced by the child canvas and frame. This feature is clearly useful for navigation aides such as visual hyperlinks and bookmarks. Interestingly, it can also be a convenient feature for other types of tools. For example, entering a magnifying glass is a convenient shortcut for increasing the scale of the canvas (magnifying the main view). Conversely, entering an overview is a simple way to demagnify the view rapidly. Entering a filter/transformer or coordinated view tool offers a convenient method of converting the main view into the alternative one shown inside the portal tool.

When the user enters a portal, to maintain the user's orientation it may be helpful for the system to perform an animated transition from the current interface state to the final state resulting from entering the portal. There are many ways to execute this animation, and we propose three alternatives based on nested space-scale diagrams here. One alternative is to deactivate both latches and navigate the parent frame to the position in which its four corners meet the four great rays of the portal frame pyramid, following one of the pan-zoom trajectories proposed in [FB95] such as a hyperbolic trajectory. The effect seen by the user of navigating along this path is that of "zooming in" on the portal until it fills the entire viewer window. A second alternative is to deactivate both latches and enlarge the portal frame gradually until its four great rays reach the four corners of the parent frame, causing the portal to appear to grow progressively until it fills the entire viewer window. In both these alternatives, after the animation has been performed, the final step is to replace the parent canvas and frame by the child canvas and frame, which will have the effect of deleting the portal outline from the viewer window perimeter and completing the seamless transformation from parent view to child view.

The previous two approaches, which align the parent and portal frames by animating the motion of one of them with both latches deactivated, can be used to animate the entrance into any portal. The following third approach may be the most effective in maintaining user orientation during the entrance into a recursive portal, or more generally any portal to another canvas with equivalent coordinates, such as a filter. This approach aligns all three frames and proceeds in two phases: one to align the portal and child frames, and a second to bring the parent frame into alignment with the other two frames. In the first phase, the parent-portal latch is activated while the parent frame is displaced (along with the latched portal frame) using a smooth animation to a new location in which the four great rays of the latched portal frame intersect the four corners of the child frame, as illustrated in the transition from Figure 5 to Figure 6. (A pan-zoom trajectory from [FB95] such as a hyperbolic trajectory can be used for this phase.) In the second phase, both latches are deactivated and the parent frame is moved to the position of the child frame, again using a pan-zoom trajectory from [FB95]. At the end of the second phase, the child frame replaces the parent frame, now at the same position, and for non-recursive portals the child canvas replaces the parent canvas, completing the transition. The overall effect of this approach seen by the user of the first phase is that the parent canvas is panned and zoomed to align the outer view to the inner one like a jigsaw puzzle. In the second phase, the main viewer "zooms in" on the portal until it fills the entire viewer window.

6 Summary and Future Work

In this paper we have described a spatial model for nested multiscale interfaces that we constructed based on experience with spatial metaphors in a prototype data visualization system. Our model uses space-scale diagrams and a physical analogy of rods piercing plates to describe the behavior of portals in response to user actions such as panning, zooming, and repositioning the portal. We also provided a taxonomy of tools that can be constructed using portals that obey our spatial model, classifying them by tool type and programmability by users. (To accompany our spatial model and portal tool taxonomy, we created an online demonstration program [Ols02], which may aid in understanding how the portal tools in our taxonomy behave in response to user actions.) Finally, we used our model of spatial relationships to propose techniques for smoothly animating the transition from the outer view to the inner view displayed inside a portal, helping to maintain user orientation during navigation between nested worlds.

The design of animated transitions between nested worlds serves as an example of how our formalisms might be used in practice. In general, it is our belief that the formal model presented here can be of significant benefit to researchers, designers, programmers, and users of nested multiscale interfaces. Researchers can use our framework as a basis for reasoning about properties of portal tool configurations including spatial dependencies and programmability options. Also, our taxonomy of possible behaviors can assist designers faced with the nontrivial tasks of deciding which of the 32 recommended behaviors to make available and how best to offer control over the tool type and programmability options to end-users. In addition, our formal model can serve as a specification for implementors of nested user interfaces. Finally, our physical analogy of plates and rods along with our interactive demonstration program may be useful in documenting the behavior of nested interface components for end-users.

As future work we plan to refine our taxonomy to allow frames in our model to be partially programmable, in addition to the existing options of full programmability and strictly non-programmability. For example, with a magnifying glass for novice users it may be appropriate to allow programming of the magnification factor, but at the same time require that the offset between the portion of the outer canvas occluded by the magnifying glass and the magnified portion displayed inside it remain fixed at zero. Doing so would require making the child frame partially programmable, since both the magnification factor and offset are encoded in the child component of the interface state. We believe that our notion of interface state, overall model, and taxonomy can be extended in a straightforward manner to accommodate partial programmability of frames.

Acknowledgments

We thank the other members of the Tioga DataSplash research group at UC Berkeley for many useful discussions over the years: Alexander Aiken, Michael Chu, Vuk Ercegovac, Mark Lin, Mybrid Spalding, Michael Stonebraker, and Allison Woodruff. We also thank Chris Stolte for helpful feedback. This research was supported by a Chambers Stanford Graduate Fellowship.

References

[AA99] G. L. Andrienko and N. V. Andrienko. Interactive maps for visual data exploration. *International Journal of Geographic Information Science*, 13(4):355–374, June 1999.

[ACSW96] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. Tioga-2: A direct manipulation database visualization environment. In *Proceedings of the 12th International Conference on Data Engineering*, pages 208–217, New Orleans, Louisiana, February 1996.

[AW95] C. Ahlberg and E. Wistrand. IVEE: An information visualization and exploration environment. In *Proceedings of the First Information Visualization Symposium*, Atlanta, Georgia, October 1995.

[Bel00] Belmont. CrossGraphs: Multidimensional graphical reporting and data visualization. White paper, Belmont Research, Inc., 2000.

[BGS99] T. Barclay, J. Gray, and D. Slutz. Microsoft terraserver: A spatial data warehouse. White paper (msr-tr-99-29), Microsoft Corporation, 1999. `http://research.microsoft.com/scripts/pubs/view.asp?TR_ID=MSR-TR-99-29`.

[BHP+96] B. B. Bederson, J. D. Hollan, K. Perlin, J. Meyer, D. Bacon, and G. W. Furnas. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. In *Journal of Visual Languages and Computing*, volume 7:1, pages 3–31, March 1996.

[BHS+96] B. B. Bederson, J. D. Hollan, J. Stewart, D. Rogers, A. Druin, and D. Vick. A zooming web browser. *SPIE Multimedia Computing and Networking*, 2667:260–271, 1996.

[BSP+93] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of the ACM SIGGRAPH Computer Graphics Annual Conference Series*, pages 73–80, Anaheim, California, August 1993.

[CMS98] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Information Visualization: Using Vision to Think*, pages 285–286. Morgan-Kaufmann, San Francisco, California, 1998.

[Cora] Microsoft Corporation. MS PowerPoint.

[Corb] Microsoft Corporation. MS Word.

[DKR97] M. Derthick, J. A. Kolojejchick, and S. F. Roth. An interactive visualization environment for data exploration. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 2–9, Newport Beach, California, August 1997.

[FB95] G. W. Furnas and B. B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 234–241, Denver, Colorado, May 1995.

[LRB+97] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. DEVise: Integrated querying and visual exploration of large datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 301–312, Tucson, Arizona, May 1997.

[MPBH95] J. Meyer, K. Perlin, B. B. Bederson, and J. Hollan. Two document visualization techniques for zoomable interfaces. Technical report, University of Maryland, 1995. `http://www.cs.umd.edu/hcil/pad++/papers/unpub-95-docvis/index.html`.

[Ols02] C. Olston. Portal demonstration applet, October 2002. `http://www.db.stanford.edu/~olston/portaldemo`.

[OW00] C. Olston and A. Woodruff. Getting portals to behave. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 15–25, Salt Lake City, Utah, October 2000.

[PF93] K. Perlin and D. Fox. Pad: An alternative approach to the computer interface. In *Proceedings of the 20th International Conference on Computer Graphics and Interactive Techniques*, pages 57–64, Anaheim, California, August 1993.

[PM99] K. Perlin and J. Meyer. Nested user interface components. In *Proceedings of the Twelfth Annual ACM Symposium on User Interface Software and Technology*, pages 11–18, Asheville, North Carolina, November 1999.

[SFB94] M. C. Stone, K. Fishkin, and E. A. Bier. The movable filter as a user interface tool. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 306–312, Boston, Massachusetts, April 1994.

[STH02] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. In *Proceedings of the Eighth IEEE Symposium on Information Visualization*, Boston, Massachusetts, October 2002.

[WOA+01] A. Woodruff, C. Olston, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding, and M. Stonebraker. DataSplash: A direct manipulation environment for programming semantic zoom visualizations of tabular data. *Journal of Visual Languages and Computing, Special Issue on Visual Languages for End-user and Domain-specific Programming*, 12(5):551–571, October 2001.