B. BELL, S. FEINER, AND T. HÖLLERER

MAINTAINING VISIBILITY CONSTRAINTS FOR VIEW MANAGEMENT IN 3D USER INTERFACES

Abstract. We present an approach to user interface design based on *view management,* in which the system and user manage graphical attributes of the user interface's components to maintain desired visual relationships. For example, we may wish to avoid having less important objects occlude more important ones. In 3D environments, this can be accomplished by having the system maintain visual constraints on the 2D projections of selected objects in the view plane to determine the objects' properties on the screen, such as position, size, and transparency. We describe a view-management mechanism that addresses dynamically changing visibility relationships among moving objects, and show how to apply it to 3D user interfaces. We focus on augmented reality systems, in which users wearing head-tracked, head-worn displays can view graphics overlaid directly onto the real world. To demonstrate some of the different ways in which the system and its users can exert control over what is seen in augmented reality, we use examples from our view-management testbed. These examples include automatically laid out annotations that provide desired information about the surrounding environment, and a hand-held tracked physical display and head-tracked virtual situation-awareness aid that respond to user control and interact with the annotated environment.

1. INTRODUCTION

A key part of user interface design involves determining the positions and sizes of the objects that the user sees, a task often known as *layout*. In 2D user interfaces, the geometry of 2D objects directly determines their geometry on 2D displays, and their assignment to layers directly determines which of a set of overlapping objects obscures the others. In contrast, in 3D user interfaces, 3D objects are viewed from one or more viewpoints, and projected onto 2D view planes that are mapped to 2D displays. Even in a static 3D scene, changing the viewpoint from which objects are seen can change the positions, sizes, and visibility relationships of their projections. Thus, the positions and sizes of the objects' 2D projections and their visibility relationships depend on both the objects and the viewpoints.

We use the term *view management* to refer to the decisions that determine the spatial relationships among the projections to obey some desired set of constraints. For example, some objects may be sufficiently important to the tasks being performed that their visibility should be guaranteed, while others may need to be rendered at some minimum size or preferred aspect ratio to improve their legibility. Furthermore, there are many situations in which the viewpoint or the geometry of certain objects may be dynamic, yet externally determined, and relatively unpredictable. For example, the viewpoint may be constrained to that of a head-tracked display, and some of the objects may be controlled by a simulation or may

Published in Stock, O. and Zancanaro, M. (eds.) Multimodal Intelligent Information Presentation (Text, Speech and Language Technology, Vol. 27), Kluwer, 2005.

be part of a changing physical world viewed through the see-through display of an augmented reality system.

In this chapter, we address how view management can be performed in interactive 3D user interfaces. We present our implementation of a view-management module that we have developed to explore real-time specification and maintenance of constraints on the projections of 3D objects on the 2D view plane. Our view-management module provides the ability to add user interface components that can interact with these constraints, to give developers the ability to add interactive controls that use view management. The interface for the view management module is used to specify constraints on objects that are tagged to indicate their properties, such as position, scale, transparency, and inter-object spatial relationships that should be maintained.

To solve 3D visibility constraints, we use interactive visible-surface determination algorithms to compute efficient approximations of the objects' projections, based on upright 2D rectangular extents. To compute placements that avoid occlusion, we apply the same algorithms to determine those portions of the view plane where objects are not projected (i.e., the empty space). We use these representations to develop a variety of view-management strategies, including ones

Figure 1. View management in an augmented reality meeting scenario (imaged through a tracked, optical see-through, head-worn display).

that take into account decisions made during previous frames to minimize frame-to-frame visual discontinuities.

Figure 1 demonstrates how we have applied our view-management module to a collaborative augmented reality environment (Arthur et al., 1998; Billinghurst et al., 1998; Butz et al., 1999; Szalavari et al., 1998). It is photographed through an optical see-through head-worn display, in which the user directly views the real world through optics, which also reflect graphics rendered on a miniature LCD. One user is sitting across from another, discussing a virtual model of the Columbia campus located between them. In response to a request from the user whose view is displayed, all virtual buildings have been annotated with their names. Each label is scaled within a user-selectable range and positioned automatically. A label is preferentially placed within a visible portion of its building's projection. If there is no space to accommodate a label legibly within that projection, it is instead placed near the projection, but not overlapping other buildings or annotations, and is connected to its building by an arrow. Additional annotations in the scene include a meeting agenda (shown at the left), a virtual copy of a building that one of the users has created, and popup information related to that building (both shown at the right). The projections of these annotations avoid other objects, including the visible user's head, to maintain a clear line of sight between users.

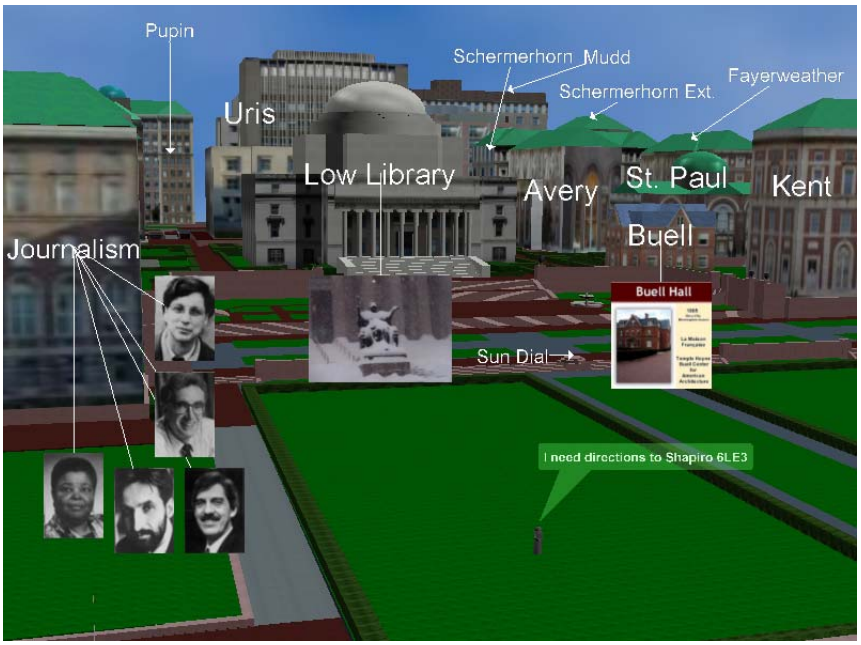

Figure 2. Screen shot of a 3D campus information visualization system.

Figure 2 shows a simple interactive 3D information visualization system that allows users to inquire about buildings on our campus. Visible buildings are initially labeled to show their identity, and to convey what is selectable. When a user selects a building, more information about it is displayed and placed close to the building's visual projection with a leader line back to the building to convey the information's relationship to it. A virtual avatar is shown in the model with a comic-strip speech balloon annotation, constrained at a fixed position relative to the projection of the avatar's mouth. All annotations are laid out so they do not overlap each other or any of the buildings' projections. Automated placement of these annotations by our view-management module ensures that important objects in the 3D scene remain unobscured when providing the user with useful information.

Placement of the annotations in these examples is personalized for each individual view. The personalized annotations appear as local variations of each view in the environment, similar to the approach used in other distributed graphics packages (Hesina et al., 2000; MacIntyre & Feiner, 1998).

In the remainder of this chapter, we describe related work in Section 2. Section 3 provides a detailed description of how we perform automated view management by collectively controlling the visibility and spatial layout of objects on the view plane, instead of indirectly modifying the properties of individual objects, as in typical 3D applications. Then, in Section 4, we extend this approach to address user-driven view management, showing how user-controlled physical objects, including a hand-held tracked physical display and a head-tracked virtual model can participate in our automated view management system. Thus, we show how view management can be accomplished jointly by the system and user working together to determine what is seen. Finally, we present our conclusions and future work in Section 5.

2. RELATED WORK

Researchers have often labeled and annotated objects in 3D virtual and augmented environments without taking into account visibility constraints (Fairchild et al., 1988; Feiner, MacIntyre, Haupt et al., 1993; Mori et al., 1999; Starner et al., 1997; Sutherland, 1968; You et al., 1999). Static 3D illustrations have been annotated using a generate-and-test approach using some visibility calculations in non-real time applications with automatic control of placement coupled with manual corrections (Rist et al., 1994). 2D graph layout algorithms (Battista et al., 1999) and 2D label placement algorithms for point, line, and area features on maps and graphs (Christensen et al., 1995) are closely related to the view-management tasks in which we are interested. Approaches to labeling point features typically explore a set of candidate positions arranged in a circle around each feature. An objective function rates a solution's goodness, based on factors such as the overlap of labels with features and other labels, and the location of labels relative to their features. Christensen, Marks, and Shieber show that an exhaustive search approach to labeling point features is NP-hard. As described in Section 4, our work uses greedy, non-optimal algorithms for positioning objects (e.g., labels or other annotations)

near points or areas. Our algorithms can efficiently determine the set of areas in which an object can be placed, while avoiding overlapping other objects and maintaining desired spatial relationships with selected objects.

It is also possible to avoid some view-management decisions by limiting the number of objects being displayed. For example, dynamic queries can allow users to filter information in a 2D database display (Ahlberg et al., 1992), while information about a user's objectives and proximity to other objects have been used to filter the 3D information displayed in an augmented reality systems (Julier et al., 2000). While these techniques are quite useful, they do not address the need to display related objects near each other, to avoid occluding objects, or to ensure visibility for specified important objects.

A number of researchers have developed approaches for automatically avoiding 3D occlusion of selected objects. Some of these rely on controlling the viewpoint (He et al., 1996; Phillips et al., 1992), which is not possible in head-tracked environments in which the viewpoint is attached to a real user's head. Others utilize illustrative effects, such as cut-away views and transparency (Feiner & Seligmann, 1992) or line style (Feiner, MacIntyre, & Seligmann, 1993; Kamada & Kawai, 1987), without altering object geometry. Visual access distortion (Carpendale et al., 1997) moves objects away from the user's line of sight to a selected focus point by displacing each object perpendicular to the line of sight by a function of the magnitude of the object's distance from the line of sight. However, this approach does not actually guarantee the visibility of an object at the focus point: the size of an object's projection is not taken into account when determining how far to move it (e.g., a large object may still occlude the focus point after the move), and the summed displacements for an object that lies between lines of sight to multiple focus points may not move the object in a useful way (e.g., the projection of an object that lies along the vector average of two lines of sight may not move). In contrast, our view-management approach can select positions for objects that guarantee desired occlusion relationships with other objects are maintained.

3. OBJECT PROPERTIES AND CONSTRAINTS

All the objects in our scenes have properties, some of which may be marked as *controllable* or *constrained* by the user, the view-management module, a tracker, or other components (e.g., a simulation or user interface). This means that there can be multiple sources of change, and these sources can control or constrain different properties of the same object. The view-management module uses the controllable properties to help satisfy the constraints. We support a set of constraints that control the view plane layout (currently visibility, position, size, and priority) and transparency of objects in a 3D scene. These constraints can be set explicitly by user interface components or by the user, or they can be defined to change based on other properties or relationships.

Visibility constraints specify occlusion relationships on the view plane: those objects that a given object should not occlude, and those objects that it is allowed to

occlude. Examples of the use of visibility constraints illustrated in this chapter include:

- Pavement and grass patches in the campus scene can always be occluded by other objects.
- Labeled objects can be occluded by certain other objects at certain times. Objects can be overlaid by their own labels, but not by labels for other objects.
- The collaborating user's face may not be occluded by any other objects.

Position constraints specify the minimum and maximum distance to be maintained from:

- Other objects: For example, the "speech balloon" of Figure 2 is constrained to be above and near an area feature (the avatar's mouth).
- A point, area, or volume in a coordinate system, which may be screen-stabilized (relative to the user's head position/orientation) or world-stabilized (relative to the world).
- The viewpoint: For stereo viewing, this distance will control how much the user's eyes need to converge to fuse the two separate images together.

Size constraints specify a range of possible sizes. For example, labels are associated with a font size range, with preference towards the high end of the range when possible.

Priority constraints specify priorities for objects. This allows the system to determine the order in which objects are included in the image, so that less important objects can be elided if adding them will violate other constraints.

Transparency constraints specify a range of object transparency values. For example, transparency can be modified to minimize the consequences of occluding other objects.

4. AUTOMATIC SYSTEM-DRIVEN VIEW MANAGEMENT

Computing the visible portions of the projections of objects on the view plane is essential for satisfying most of the constraints described. Figure 3 shows a pipeline that specifies the order in which computations are done for view management. To precisely maintain the specified visual constraints, this pipeline should get executed for each frame rendered. However, since some of these computations can take a considerable amount of time to compute, asynchronous execution of the pipeline for real-time applications, as in the Decoupled Simulation Model (Shaw et al., 1993), can preserve interactive rendering frame rates at the expense of view management accuracy. (These errors might be preferable depending on the user's preference or task.) In this section, we discuss each pipeline step and give examples of how it is performed in our implementation.

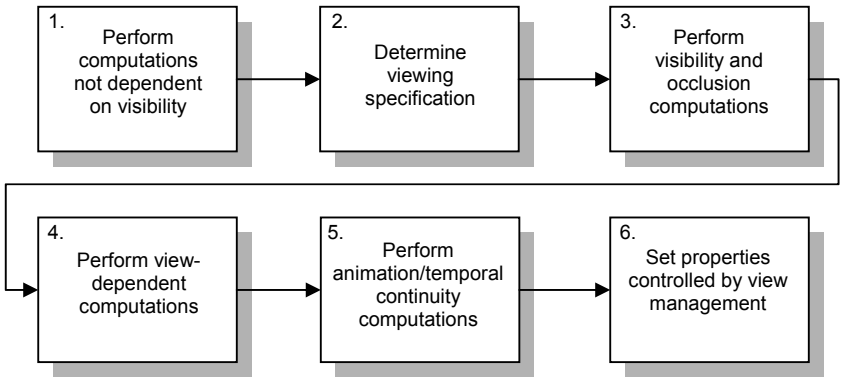

Figure 3. View Management Pipeline

4.1. Computations not Dependent on Visibility

Computations that are not dependent on the viewing specification can happen either at the beginning of the pipeline or asynchronously. Some applications, such as ones that use head-worn displays, must render graphics based on real-time tracker input. To minimize the system's response time (or error caused by asynchronous computation latency), we want to minimize the time between reading the tracker data (i.e., to determine the viewing specification in step 2) and setting object properties (step 6). To do this, we execute all computations that are not dependent on the viewing specification outside of these steps. These computations can include most types of logic in a graphics system, such as layout, content management, and interaction. Also, data structures that are needed farther down the pipeline can be initialized at this time.

4.2. Viewing Specification Determination

For each 3D object in the scene, we need to determine projection parameters that include viewpoint (two for stereo), orientation, and other viewing parameters (e.g., field of view and clipping planes). For a physical simulation, all objects in the same coordinate system will typically have the same relative viewing specification. Depending on the type of graphical application, determining the viewing specification for each frame can be different. We want to support applications in which the viewing specification changes interactively and unpredictably, as well as applications in which the view follows a predetermined path. Some real-time applications determine the user's viewpoint and orientation by using input from 3D trackers, along with transformations based on the position and orientation of the physical trackers and the display relative to the user. Other real-time 3D

applications are controlled using 2D direct manipulation devices, such as a mouse. Predetermined applications, such as video and animation playback, can be manually planned or use automated assistance (He et al., 1996; Phillips et al., 1992). Determining the viewing specification may involve screen-space computations to automatically choose camera placement.

The situation-awareness aid we describe in Section 5 determines a viewing specification for a world in miniature that depends on the user's head pitch, which is computed from the viewing specification for the physical world. In this augmented reality example, two viewing specifications, one for the physical world and one for the world in miniature, are computed from tracker input for rendering.

4.3. Visibility and Occlusion Computations

After determining the viewing specification, the constrained objects' visibility, occlusion and depth values are computed to help satisfy the constraints discussed in Section 3. To satisfy these constraints efficiently, we use a 2D space representation (Bell & Feiner, 2000) to encode the visible portions of the projections of 3D objects on the view plane and a second instantiation of this 2D space representation to encode the complementary portions of the view plane on which no projections lie (i.e., the empty space). Each such representation of an arbitrary, possibly multifaceted, spatial region consists of a list of rectangles that are axis-aligned, are possibly overlapping, and are the largest such rectangles that lie wholly inside that region. For example, Figure 4(a) shows four images of a view plane populated by a single (dark grey) rectangle, which represents the projection of a single 3D object. Each image shows one of the four (light grey) rectangles that together comprise the representation of the remaining empty space. These four light grey rectangles are the largest axis-aligned rectangles that fit wholly within the portion of the view plane that does not include the dark grey rectangle; that is, none of these largest space rectangles can get any larger and still remain completely inside the region that it is supposed to represent.

Suppose we wish to place an object within either region. If that object can be represented by an axis-aligned rectangle, it will fit within a region only if its rectangle fits wholly within one of the rectangles that make up that region's representation. Furthermore, if the object can be translated, the union of all positions it can occupy while wholly inside at least one of the region's rectangles represents all valid positions it can occupy within the entire region. Thus, by searching through the rectangles representing a region, it is possible to find placements inside that region that match specified containment constraints (Section 4.4).

To satisfy some constraints, such as avoiding occlusion, we can search for a position for an object within the empty-space representation. For other constraints, we need to compute the visible-space representation of one object, or of a combination of objects. We have used two alternative visible-surface-determination algorithms to generate these representations: one based on a Binary Space Partitioning Tree, and one based on a hardware-accelerated z-buffer.

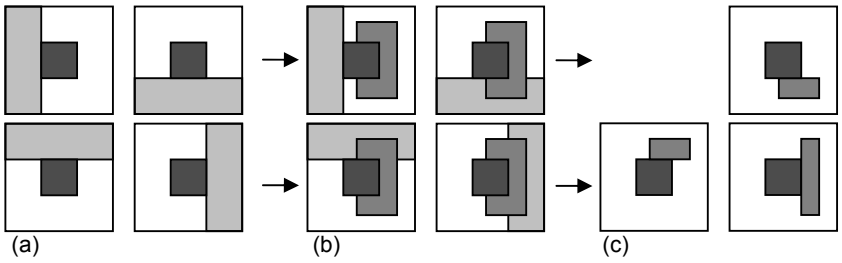

Figure 4. Adding a new object to the view-plane representation. (a) original object (dark grey), shown separately with each of the four original largest empty spaces (light grey) (b) New object extent (medium grey) is added. (c) Resulting visible largest spaces of new object extent (medium grey)

4.3.1. Visibility Determination using a Binary Space Partitioning (BSP) Tree

We can use the Binary Space Partitioning (BSP) Tree algorithm (Fuchs et al., 1980) to determine the visibility order efficiently for an arbitrary projection of a scene. A BSP tree is a binary tree whose nodes typically represent actual polygons (or polygon fragments) in the scene. Because we need to determine the visibility order for objects, rather than for the polygons of which they are composed, our BSP tree nodes are defined by planes that separate objects, rather than planes that embed object polygons. We choose these planes using the heuristics of (Thibault & Naylor, 1987). Although BSP trees are often used for purely static scenes, dynamic objects can be handled efficiently by adding these objects to the tree last and removing and adding them each time they move (Chrysanthou & Slater, 1992). We traverse the BSP tree in front-to-back order relative to the viewpoint to find the visible portions of the scene's objects.

For each node obtained from the BSP tree in front-to-back order, the new node's upright extent is intersected with the members of the current list of largest empty-space rectangles. This can be accomplished efficiently because we maintain the largest empty-space rectangles in a 2D interval tree (Samet, 1990), allowing an efficient window query to determine the members that actually intersect the extent. (The intersection operation itself is trivial because all rectangles are axis-aligned.) The intersection yields a set of rectangles (not necessarily contiguous), some of which may be wholly contained within others; these subset rectangles are eliminated. The result is a set of largest rectangles whose union is the visible region of the new node. For example, Figure 4(b) shows the intersection of a medium grey rectangle (the upright extent of the second BSP tree node to be processed in this case) with the four light grey largest empty-space rectangles. The resulting three medium grey rectangles, shown in Figure 4(c), represent the visible portion of the original medium grey rectangle.

Some objects are represented by a single BSP tree node, while others are split across nodes. (Objects may be split during BSP tree construction, or prior to BSP tree construction to better approximate a large object by the upright extents of a set of smaller objects.) For example, if a single object were split in two by a BSP tree plane, it would be represented by two sets of largest visible-space rectangles, one per node. Another rectangle being compared with the largest visible-space rectangles in both sets might fit in none of them, even though it might fit in one of the largest visible-space rectangles representing the original unsplit object, resulting in a false rejection. Therefore, the visible rectangles of all of an object's nodes must be coalesced to create the correct list of largest rectangles for the object (Bell et al., 2001).

4.3.2. Visibility Determination using a Z-buffer

Alternatively, we can compute the 2D spatial representation of visible regions of objects in a 3D scene using a *z*-buffer algorithm (Catmull, 1974). By rendering each object in a distinct color with the current viewing specification, we can use the frame buffer as an *object buffer* (Atherton, 1981), in which each pixel's color encodes the object visible at that pixel.

We use a sweep algorithm to convert the resulting pixel representation to the spatial representation needed to satisfy our constraints. Since it is likely that adjacent pixels will have the same value, we evaluate each pixel, starting at one corner of the buffer, across the largest dimension of the image, keeping track of the last position at which the pixel value has changed and the current object value. Once a change is detected or the end of the scan line is reached, a rectangle is created from the last to the current position (with unit height) and combined into the current object's visible-space representation. The current object and last position are then reset, and the algorithm continues. Since the only rectangles that can be adjacent to a rectangle on the current scan line are those combined from the previous scan line, a 2D interval tree of all rectangles from the previous scan line is kept for fast lookup. Any rectangle that is not a result of a combine operation from the last scan line is considered part of the final result and does not need further processing. The set of largest visible space rectangles for each visible region consists of all resulting rectangles, including the rectangles that have been combined on the last scan line.

Note that we must be careful to avoid the problems that can occur when two adjacent polygons in the same object fail to share a vertex on a shared edge, potentially causing missed pixels (Foley et al., 1996). This can result in a large area being improperly split into a set of smaller ones. To prevent this, we add the necessary extra vertices to polygons in our model.

4.3.3. Performance and Accuracy

Since most current graphics hardware implements the *z*-buffer algorithm, the time needed to compute the pixel representation is low. However, the conversion to our representation is more computationally intensive. Generating the object buffer at a

lower resolution than used for rendering the user interface accelerates the algorithm at the expense of accuracy.

In the BSP tree approach, accuracy can also be traded for performance. Since objects are approximated by the upright rectangular extents of their BSP tree nodes, better approximations can be produced if more nodes are generated at the cost of increased computation.

4.4. View-Dependent Computations

The resulting visibility representations, consisting of the largest rectangles of single objects, multiple objects, and empty spaces (i.e., background), can support a variety of layout strategies. Queries on these visible regions can be designed for choosing appropriate positions and sizes for new objects within these areas. For placing text, this is equivalent to map area feature labeling (Roessel, 1989) in a 3D environment in which objects can partially occlude each other; in this situation, we can place the label relative to the visible portion of the related object. The visibility representations also allow proximity queries that can constrain the resulting position of an object to be inside a region and closest to a point or another projected object. These queries are satisfied by searching the region's list of largest visible rectangles to find a resulting rectangle that best fits the criteria needed, similar to the queries of (Bell & Feiner, 2000).

We describe here some examples of strategies used for satisfying specific constraints for application tasks such as labeling, annotating, and placing other information. The results from these computations can be used to place objects directly or as input to animation computations (Section 4.5) that produce the final placement.

There are many cases in 3D user interfaces in which the application needs to communicate information to the user. For example, Figure 5 shows an augmented reality environmental browsing mode in which the system tries to label a number of objects that have been determined to be of interest to the user, making sure that the labels are not ambiguous. In this case, we use the two-tiered approach that was applied to Figures 1–2, creating both internal and external labels, and ensuring that no label occludes any part of an object except the object it labels. In contrast to Figure 1, Figure 5 is imaged using the video see-through display, in which video mixing hardware is used to combine a camera view of the real world with synthesized graphics.

4.4.1. Internal Label Placement

For each object to be labeled, we determine the largest visible rectangle internal to the object's projection that can contain the largest copy of the object's label, given a user-settable font and size range, and taking into account an additional buffer around the label to keep adjacent labels from appearing to merge. This buffer can also help control the amount of the related object's projection that is blocked by the label if the object has a high priority. For efficiency, we approximate the dynamic size of a

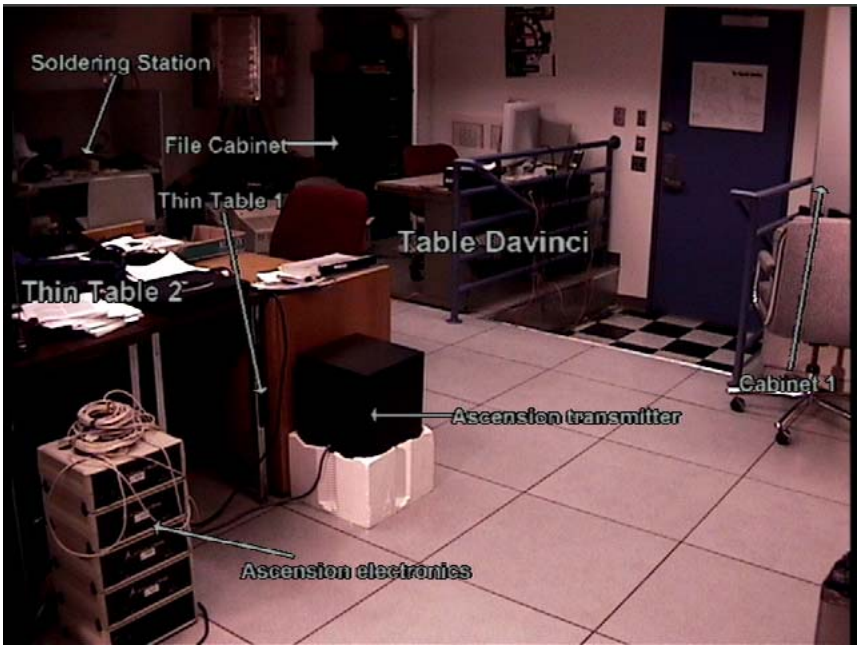

Figure 5. Browsing an augmented reality environment (imaged through a tracked video see-through display).

label using its aspect ratio at a certain font size, texture map it to a polygon, and scale the polygon based on the result of the query.

4.4.2. External Label Placement

If no internal rectangle for an object is large enough, the object will not be labeled internally, but could instead be labeled externally. External labels can be processed in any desired order. We currently use the front-to-back order; however, since the allocation algorithm is greedy, if an importance metric were available for labels, we could sort on it instead. To lay out an external label, the system queries the empty-space representation for a rectangle that can contain the label within a user-settable range of sizes, and within a user-settable distance from the object. If the label is allocated within the empty space, it will neither be occluded by nor occlude any other object. If no such rectangle can be found, then the label is not allocated.

Since external labels are potentially ambiguous, we also generate a leader-line arrow from the label to the interior of its object. Note that many classical map labeling algorithms base their label-placement quality estimate on whether the label can be visually identified easily with only a single feature (Imhof, 1975). Providing

Figure 6. Image taken directly through an optical see-through head-worn display of restaurant and related popup information. Note: Since the construction vehicle and cable spool on the right are not included in our environment model, the system does not constrain the popup to avoid overlapping them. The brightly lit spool appears to obstruct the virtual popup because it is much brighter than the image on the optical see-through display.

the leader line helps mitigate possible misidentifications.) Because internal labels occlude parts of the object that they label, we also support tagging an object to indicate whether or not certain parts should be internally labeled (e.g., internal labels may be suppressed for faces). Each external label is added to the view-plane representation as it is created, so that objects added later do not occlude it.

4.4.3. Other Placements

Applications may need to show additional information about objects in the scene (e.g., Figures 1–2 and 5). This information can be placed external to related objects, much like external labels, possibly with different size and aspect ratio constraints.

Some information, such as the agenda located on the left side of Figure 1, does not refer to any objects that are visible in the scene. To place the agenda on the screen so that it doesn't overlap any important objects, the empty-space region is queried after all other annotations are placed. There are also situations, such as the augmented reality neighborhood restaurant guide of Figure 6, where there is not

enough room to place the information. In this case, we detect the limited space by the negative results of external placement. Another query is used to place the popup information so that it avoids overlapping the restaurant, but could overlap other objects, such as the building above it.

4.5. Animation/Temporal Continuity Computations

Thus far, we have described our system without taking into account temporal continuity. However, if the layout of objects in sequential frames is computed independently, discontinuous changes in object position and size occur, which may be annoying in some applications. Therefore, in some real time applications, we need to take into account decisions made during previous frames when determining current placement. First, we introduce hysteresis in the discrete state changes that can take place. We also use a technique that is similar to traditional keyframe animation (Lasseter, 1987): comparisons between the results of the discrete queries (Section 4.4) and additional queries specific to animation (Section 4.5.2) automatically determine the keyframes in real time for determining placement destinations, while interpolation computes the animation between the keyframe placements (the "inbetweening"). Instead of introducing movement, this technique results in the annotations being continuously rendered at the keyframes for positional stability, while inbetween frames are rendered minimally (only when movement between keyframes is necessary).

Based on informal user feedback, when the viewing specification is controlled by a mouse, animation is typically not preferred—when the user stops controlling the view, the viewing specification stops changing, but the annotations would keep moving if animation were used. In contrast, when the viewing specification is controlled by a head tracker, it is always changing, if only slightly, potentially resulting in drastic changes in the results of our layout queries between frames. Thus, to avoid jitter and excessive movement of annotations, it is necessary to provide animation and temporal continuity.

4.5.1. State Hysteresis

There are several situations in which objects may change state, resulting in a discrete visual change, such as from an internal label to an external one, or from being displayed to not being displayed. Some user interfaces use three constraints on the size of an object being displayed: minimum size, maximum size, and preferred size (e.g., Java AWT (Gosling et al., 1996)). As shown in the state diagram of Figure 7, we borrow this approach, modifying the definition of minimum size by defining both an absolute minimum size (*absMin*) and a sustained minimum size (*min*) to make possible state hysteresis to avoid oscillation between states at size boundary conditions.

An object's visual size is guaranteed to be the sustained minimum size at least once within a settable time interval *n*. The system displays the object only when there is enough space for the sustained minimum size, and removes it when it is

below the absolute minimum size. Furthermore, if the object is already being displayed and there is only enough space for an object within the absolute and sustained minimum sizes for time > *n*, the object is removed. Otherwise, the object is displayed at the largest possible size no greater than its maximum size. The state diagram of Figure 7 employs similar tests to make an additional distinction between drawing an object inside (preferred) vs. outside another object, which we use for displaying area feature labels. The values selected for *n* and the difference between the absolute minimum and sustained minimum sizes help avoid visual discontinuities.

4.5.2. Positional Stability

To compute whether the current frame is a "keyframe" for a particular object, it is necessary to find out whether the object's position from Section 4.4, which is considered the best placement for the current frame, has moved from the object's position in the last computation. To do this, an additional operation, computed in pipeline step 4, needs to query for the closest layout to the previous layout. The same visible region is used for this query. For example, if the label was an inside label in the previous frame, the visible region of the object would be queried. However, since the annotation's related object could have moved because of view specification change (e.g., due to head movement), the last placement is not valid for finding the closest layout. Instead, we compute the change relative to the projection of the related object by transforming the last frame's placement from the 2D coordinate system of the projected object in the last frame, to the 2D coordinate

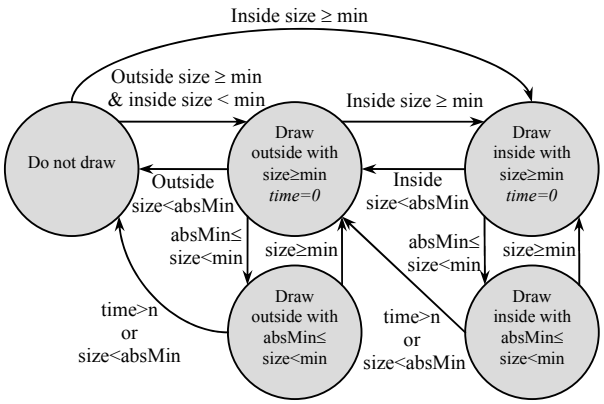

Figure 7: State hysteresis in area feature labeling. Area features are preferentially labeled inside rather than outside. Labels are promoted from left to right through comparison with min and demoted from right to left through comparison with absMin.

system of the projected object in the current frame, where the centroid of each rectangle is its own origin. The result, transformed back into pixel coordinates, is used as the parameter to query for the closest relative placement from the previous frame inside the current frame. This result is compared with the best layout and is used if both results are the same within a small threshold; if they are different, we start a timer and, if the best and closest fail to coincide after a set amount of time, we use the best position as a keyframe and reset the timer.

4.5.3. Interpolation

Once a keyframe is chosen, the current frame is determined by interpolating the layout parameters from the previous frame to the keyframe. We use linear interpolation for both position and scale, based on the time difference since the last frame. In changing from internal to external labels, we also grow or shrink the arrow.

4.6. Set Controlled Object Properties

When all view management pipeline computations have completed, the properties of the rendered objects must be updated. Although the pipeline can run asynchronously relative to rendering, as described at the beginning of Section 4, inconsistent results can occur if the rendering properties for an object do not get set before it is rendered. Therefore, all controlled object properties need to get set before rendering begins, assuming that any object can affect another object's projection.

5. USER-DRIVEN VIEW MANAGEMENT FOR AUGMENTED REALITY

In many augmented reality applications, we would like to assist the user in interacting with and obtaining information about the physical environment. In our testbed, users can control annotations and other user interface components by indirect manipulation; a mouse or other wireless device controls a menu to select filtering modes or manipulate object properties and constraints. Selecting a label reveals more information, which can present buttons or other widgets to offer additional options for obtaining additional information.

These interactions with virtual objects, although intuitive and useful, require significant user attention to control and attend to the information. We have explored how direct manipulation techniques can be used to create user interfaces that demand less user attention. Shneiderman (Shneiderman, 1983) states the essential ingredients to direct manipulation include immediate response to actions, incremental changes, and reversible effects. These techniques, when applied to augmented reality, can help the user balance their attention between the real world and virtual overlaid objects, and can interactively control what is seen virtually.

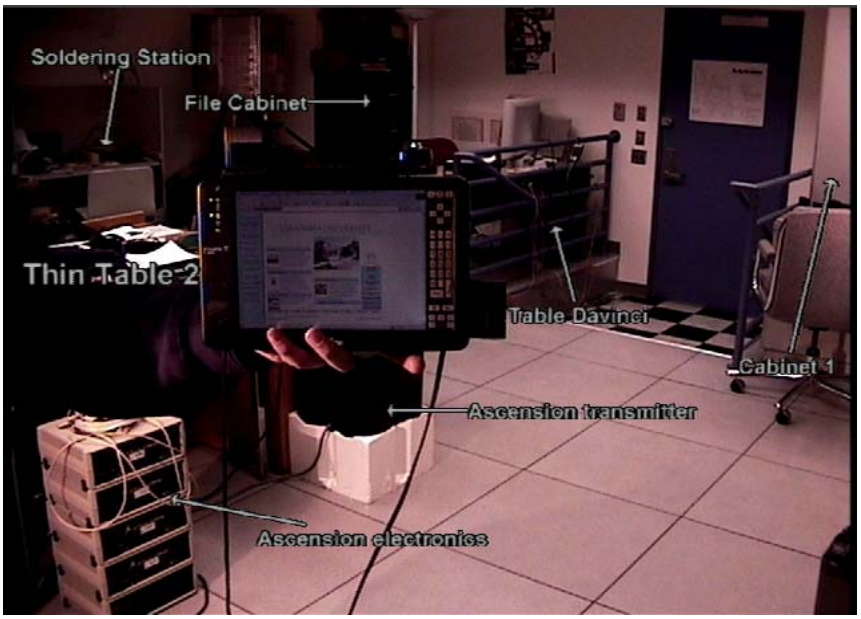

Figure 8: Tracked hand-held display takes priority over labels.

5.1. Physical Manipulation

Figure 8 shows a screen shot of a user interactively controlling a six-degree-of-freedom-tracked, hand-held computer, whose screen we would not like to have obscured by labels. To accomplish this, the screen is defined as having a higher priority than any of the objects in Figure 5, and no associated label. Consequently, since much of "Table Davinci" is obscured by the hand-held computer in Figure 8 relative to Figure 5, the "Table Davinci" label changes from a large internal label to a smaller external one; the "File Cabinet" and "Ascension Transmitter" labels and arrows move up and down, respectively, out of the hand-held computer's way; and the "Thin Table 1" label disappears.

Imagine replacing the tracked hand-held computer with a transparent tracked frame. Instead of having the system show labels for all objects that are not obscured by the hand-held computer, the system could act like a 3D magic lens (Viega et al., 1996), showing labels only for objects that fall within the frame.

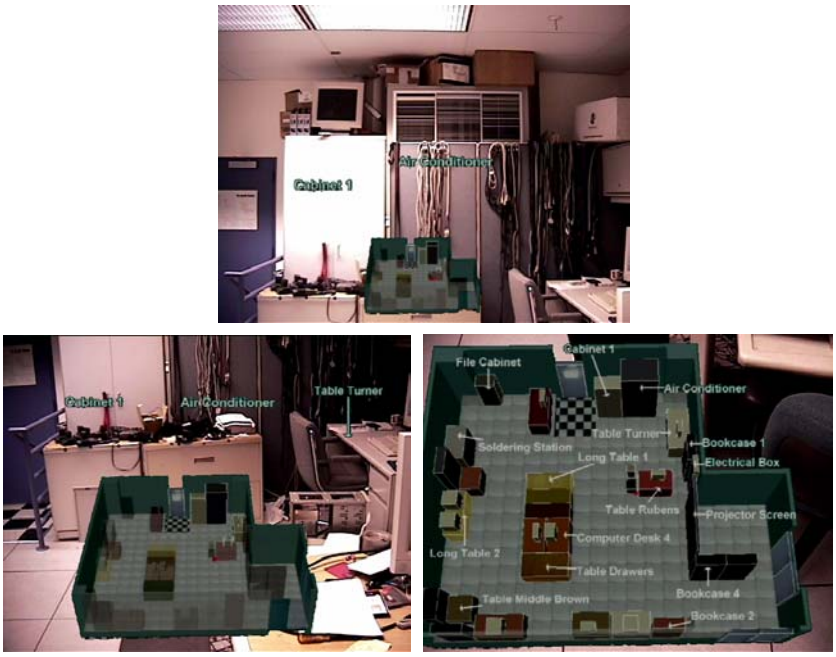

Figure 9. Annotated situation-awareness aid. (a) Looking slightly upward. (b) Looking slightly downward. (c) Looking nearly straight down.

5.2. Controlling a Situation Awareness Aid

We have developed a situation-awareness aid, based on a world in miniature (Stoakley et al., 1995), that provides the user with an overview of the surrounding environment and the ability to discover, select, and inquire about objects that may not be directly visible within that environment. The user's head pitch controls the aid's position, scale, and orientation. This makes possible user initiated view-management control: by using head movement, users can easily change their focus of attention between the real world and the virtual aid.

The aid's yaw (*y*-axis orientation) is constrained to be aligned with the environment, as in (Darken & Sibert, 1993). Figure 9 shows screen shots of the aid in our laboratory environment. In Figure 9(a), the user looks slightly upward and the unannotated aid stays close to the bottom of the viewport, pitched towards the view plane by a default angle (22.5˚) about the user's position in the aid. In Figure 9 (b), the user looks slightly downward and the aid is scaled to be slightly larger, angled slightly closer to parallel to the view plane, and translated up slightly higher in the viewport. In Figure 9(c), the user looks nearly straight down, and in response

Figure 10. Outdoor version of the annotated situation-awareness aid.

receives a zoomed-in, annotated view that is nearly parallel to the view plane, with the user's position in the aid located at the center of the viewport. Figure 10 shows a version of the aid in our outdoor augmented reality system that uses both texture-mapped aerial photography as well as 3D models of individual buildings.

The roll component of the aid's orientation is kept at zero, so that its ground plane is always parallel to the bottom of the viewport, but not necessarily parallel to the ground plane in the physical world. Note that the only effect of the user's position on the aid is to control the "you are here" marker, rendered as a red sphere, about which the aid orients (in yaw and pitch). This marker is positioned at the center of the viewport's width and at a head-pitch–determined location relative to the viewport's height.

To obtain the desired functionality, the user's head pitch is used to determine three parameters of the aid: pitch, scale, and *y* position in the viewport. Each parameter has a set of four values: min and max values and two head pitch trigger values. As described in (Bell et al., 2002), we linearly interpolate each dependent variable from min to max between the head pitch trigger values.

No parameter is influenced above zero degree head pitch. That means that as long as the user looks straight ahead or up, the aid occupies the smallest possible amount of screen space allowed by the parameters and stays near the bottom of the

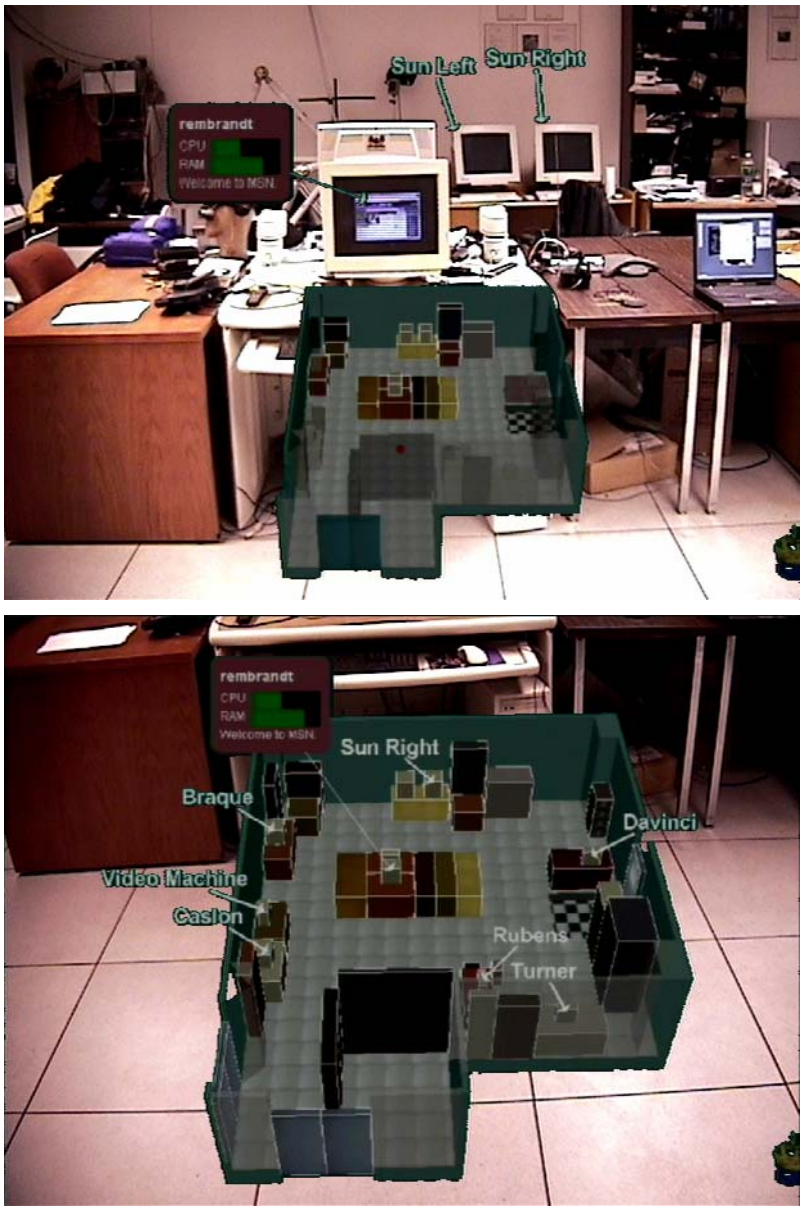

Figure 11. Transfer of annotations between the physical world and virtual aid. (a) Popup annotation provides information about the computer monitor to which its arrow points at the center. (b) As user looks down, annotations are enabled in the aid, and the popup's arrow now points at the monitor's representation in the aid.

viewport. Both the scale and the vertical position of the aid are set to reach their maximum values when the user looks down at a 45° angle.

5.3. Annotation Assistance

The aid provides us with another set of capabilities for providing the user with more information and interaction. In Figure 9(c), labels are placed relative to the projections of the virtual objects in the aid in the same way we annotate the physical objects. A threshold value (currently set at looking down 30°) determines when the labels change between annotating the real world and labeling the aid.

Popup annotations behave in the exact same way. Figure 11(a) shows an annotation reporting on a computer's CPU and memory usage, placed relative to the projection of its associated computer's physical monitor, while avoiding other important objects, including other physical objects, labels, and the aid. As the user looks down at the aid, the annotated physical monitor is no longer visible, but the popup remains displayed. As the aid scales up, its labels appear, and the popup's arrow points to the monitor's virtual representation in the aid, shown in Figure 11(b). As the user looks up, the popup is dissociated from the aid and recaptured by the physical monitor in the full-scale physical environment. Thus, popup annotations can move freely between the physical world and its scaled virtual representation. Furthermore, virtual objects in the aid can be selected to view information about physical objects that might not be currently visible in the real world from where the user is currently positioned.

6. CONCLUSIONS

We have presented a framework for view management in virtual and augmented reality applications. While many of the underlying algorithms we described may change with further development and refinement as we improve performance, add possible constraints, and increase functionality, we believe that the basic concept of view management is fundamental to creating interactive 3D applications: placing and automatically maintaining visual constraints on the visibility and spatial layout of the projections of 3D objects.

Many current applications can benefit from using view management techniques, ranging from scientific visualization to entertainment; for example, view management would be particularly applicable to labeling characters in 3D multi-player games. Although augmented reality has attracted increasing interest from academic and industrial researchers over the past decade, head-tracked, head-worn systems are at present still research prototypes. For augmented reality applications to flourish, advances will be required in hardware technology (displays and tracking), user interfaces, and societal acceptance (Azuma et al., 2001). The techniques presented in this chapter contribute to the user interface, addressing both information presentation and user interaction.

We are extending our work in three major directions. First, we have experienced some situations in which the current layout strategies induce excessive movement of annotations, such as when the user's head moves quickly, or when the system

annotates small objects. To cope with this problem, we are developing strategies in which annotations are placed in 3D but dynamically take into account visibility relationships, though less aggressively than our current techniques. One variant, similar to "billboarding," would always be oriented facing the user. Second, we plan to do usability studies to compare the effectiveness of these different techniques in a variety of tasks. Using eye tracking to determine where the users' eyes move relative to the 3D scene could help measure the effectiveness of these techniques (Tanriverdi & Jacob, 2000). Finally, we note that all our constraints are currently placed "by hand," which is quite tedious. To address this limitation, we are developing a knowledge-based infrastructure to allow us to control programmatically how constraints are used. This will make it easier to handle more complex situations, including ones in which the constraints are inferred by rules.

7. ACKNOWLEDGMENTS

This work was supported by ONR Contracts N00014-99-1-0249, N00014-99-1-0394, and N00014-99-1-0683, NSF Grants IIS-00-82961 and 0121239, NLM Contract R01 LM06593-01, and gifts from Intel, Microsoft, and Mitsubishi Electric Research Labs. Blaine Bell was supported by an IBM Graduate Research Fellowship. Ryuji Yamamoto created the campus model shown in Figures 1–2.

8. REFERENCES

Ahlberg, C., Williamson, C., & Shneiderman, B. (1992). *Dynamic Queries for Information Exploration: An Implementation and Evaluation.* Proc. ACM CHI '92.

Arthur, K., Preston, T., Taylor, R., II, Brooks, F., Jr., Whitton, M., & Wright, W. (1998). Designing and Building the PIT: A Head Tracked Stereo Workspace for Two Users. In *Second Int. Immersive Projection Technology Workshop*. Ames, IA.

Atherton, P. R. (1981). *A Method of Interactive Visualization of CAD Surface Models on a Color Video Display.* Proc. SIGGRAPH '81.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., & MacIntyre, B. (2001). Recent Advances in Augmented Reality. *IEEE Computer Graphics and Applications, 21*, 34–47.

Battista, G. D., Eades, P., Tamassia, R., & Tollis, I. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. USA: Prentice Hall.

Bell, B., & Feiner, S. (2000, November 5--8). *Dynamic Space Management for User Interfaces.* Proc. UIST '00, San Diego, CA.

Bell, B., Feiner, S., & Höllerer, T. (2001, November 11–14). *View Management for Virtual and Augmented Reality.* Proc. UIST '01, Orlando, FL.

Bell, B., Höllerer, T., & Feiner, S. (2002, October 27–30). *An Annotated Situation-Awareness Aid for Augmented Reality.* Proc. UIST '02, Paris, France.

Billinghurst, M., Weghorst, S., & Furness, T., III. (1998). Shared Space: An Augmented Reality Approach for Computer Supported Collaborative Work. *Virtual Reality, 3*(1), 25–36.

Butz, A., Höllerer, T., Feiner, S., MacIntyre, B., & Beshers, C. (1999). Enveloping Users and Computers in a Collaborative 3D Augmented Reality. In *Proc. IWAR '99 (Int. Workshop on Augmented Reality)* (pp. 35–44). San Francisco, CA.

Carpendale, M. S. T., Cowperthwaite, D. J., & Fracchia, F. D. (1997). Extending Distortion Viewing from 2D to 3D. *IEEE Computer Graphics and Applications, 17*(4), 42–51.

Catmull, E. (1974). *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Ph. D Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT.

Christensen, J., Marks, J., & Shieber, S. (1995). An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Transactions on Graphics, 14*(3), 203–232.

Chrysanthou, Y., & Slater, M. (1992). Computing dynamic changes to BSP trees. In *Computer Graphics Forum (Proc. Eurographics '92)* (Vol. 11, pp. 321–332).

Darken, R. P., & Sibert, J. L. (1993, November). *A Toolset for Navigation in Virtual Environments*. Proc. UIST '93, New York, NY.

Fairchild, K. M., Poltrock, S. E., & Furnas, G. W. (1988, May). *SemNet: Three-Dimensional Graphic Representation of Large Knowledge Bases.* Cognitive Science and its Applications for Human-Computer Interaction, Hillsdale, New Jeresy, U.S.A.

Feiner, MacIntyre, B., Haupt, M., & Solomon, E. (1993). Windows on the World: 2D Windows for 3D Augmented Reality. In *Proc. UIST '93 (ACM Symp. on User Interface Software and Technology)* (pp. 145–155). Atlanta, GA.

Feiner, MacIntyre, B., & Seligmann, D. (1993). Knowledge-based augmented reality. *Communications of the ACM, 36*(7), 52–62.

Feiner, S., & Seligmann, D. (1992). Cutaways and Ghosting: Satisfying Visibility Constraints in Dynamic 3D Illustrations. *The Visual Computer, 8*(5–6), 292–302.

Foley, J., van Dam, A., Feiner, S., & Hughes, J. (1996). *Computer Graphics: Principles and Practice, 2nd Ed. in C*. Reading, MA: Addison-Wesley.

Fuchs, H., Kedem, Z., & Naylor, B. (1980, July 14-18). *On visible surface generation by a priori tree structures.* SIGGRAPH '80, Seattle, WA.

Gosling, J., Joy, B., & Steele, G. (1996). *The JAVA Language Specification* (1st ed.). Reading, Massachusetts: Addison-Wesley.

He, L., Cohen, M., & Salesin, D. (1996). The virtual cinematographer: A paradigm for automatic real-time camera control and direction. In *Proc. SIGGRAPH '96* (pp. 217–224). New Orleans, LA.

Hesina, G., Schmalstieg, D., Fuhrmann, A., & Purgathofer, W. (2000, Decemeber 20--22). *Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics.* Proc. VRST '99, N.Y.

Imhof, E. (1975). Positioning names on maps. *The American Cartographer, 2*(2), 128–144.

Julier, S., Lanzagorta, M., Baillot, Y., Rosenblum, L., Feiner, S., Höllerer, T., et al. (2000). Information Filtering for Mobile Augmented Reality. In *Proc. ISAR '00 (Int. Symposium on Augmented Reality)* (pp. 3–11). Munich, Germany.

Kamada, T., & Kawai, S. (1987). An enhanced treatment of hidden lines. *ACM Transactions on Graphics, 6*(4), 308–323.

Lasseter, J. (1987). *Principles of traditional animation applied to 3D computer animation.* Proc. SIGGRAPH '87.

MacIntyre, B., & Feiner, S. (1998). A Distributed 3D Graphics Library. In *Computer Graphics (Proc. ACM SIGGRAPH '98)* (pp. 361–370). Orlando, FL.

Mori, T., Koiso, K., & Tanaka, K. (1999). Spatial Data Presentation by LOD Control Based on Distance, Orientation, and Differentiation. In *Proc. of Int'l Workshop on Urban Multi-Media/3D mapping (UM3'99)* (pp. 49–56). Tokyo, Japan.

Phillips, C. B., Badler, N. I., & Granieri, J. (1992, March -- April). *Automatic Viewing Control for 3D Direct Manipulation.* Proc. SIGGRAPH '92, Cambridge, MA.

Rist, T., Krüger, A., Schneider, G., & Zimmermann, D. (1994). *AWI: a workbench for semi-automated illustration design.* Proceedings of the workshop on Advanced visual interfaces, Bari, Italy.

Roessel, J. (1989). An Algorithm for Locating Candidate Labeling Boxes within a Polygon. *The American Cartographer, 16*(3), 201–209.

Samet, H. (1990). *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley.

Shaw, C., Green, M., Liang, J., & Sun, Y. (1993). Decoupled simulation in virtual reality with the MR toolkit. *ACM Transactions on Information Systems (TOIS), 11*(3), 287–317.

Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer, 16*(8), 57–69.

Starner, T., Mann, S., Rhodes, B., Levine, J., Healey, J., Kirsch, D., et al. (1997). Augmented Reality through Wearable Computing. *Presence, 6*(4), 386–398.

Stoakley, R., Conway, M., & Pausch, R. (1995, May 7–11). *Virtual Reality on a WIM: Interactive Worlds in Miniature.* Proc. CHI '95, Denver, CO.

Sutherland, I. (1968). *A Head-Mounted Three Dimensional Display.* Proc. FJCC 1968, Washington, DC.

Szalavari, Z., Schmalstieg, D., Fuhrmann, A., & Gervautz, M. (1998). Studierstube: An Environment for Collaboration in Augmented Reality. *Virtual Reality, 3*(1), 37–48.

Tanriverdi, V., & Jacob, R. (2000). *Interacting with Eye Movements in Virtual Environments.* Proceedings of the SIGCHI conference on Human factors in computing systems, The Hague, The Netherlands.

Thibault, W., C., & Naylor, B., F. (1987). *Set operations on polyhedra using binary space partitioning trees.* Proc. SIGGRAPH '87.

Viega, J., Conway, M. J., Williams, G., & Pausch, R. (1996). *3D Magic Lenses.* UIST '96, Seattle WA.

You, S., Neumann, U., & Azuma, R. (1999). Hybrid Inertial and Vision Tracking for Augmented Reality Registration. In *Proc. IEEE Virtual Reality '99* (pp. 260–267). Houston, TX.