

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 objects that are more important be occluded by objects that are less important. In 3D environments, this can involve the system maintaining 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 component that addresses dynamically changing visibility relationships among moving objects, and apply it to 3D user interfaces, focusing 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 show examples from our testbed systems. These examples include automatically laid out annotations that provide desired information about the surrounding environment, and a situation awareness aid whose display responds to the user's head movement.

1. INTRODUCTION

When displaying objects in a 3D graphical user interface (UI), the perspective projections on the 2D display are difficult to obtain before rendering. Only if the viewing specification and all objects' properties, including position, scale, and rotation relative to the view, are known, it is possible to compute the projections by using a visible surface determination algorithm. However, in interactive applications, the system and its users can change objects' properties throughout the application, including changes that can influence the projection. Each change to an object's property could, in turn, affect the visibility computations of itself and other objects. Instead of indirectly effecting what is seen on the screen by changing individual object's properties, we are interested in using visual constraints to automatically control the visibility and spatial layout of objects on the view plane. We refer to maintaining constraints for all objects in a scene and their relationships as view management. For example, some objects may be sufficiently important to the user's task that the system should guarantee visibility. Some objects might only be displayed if other related objects are visible, while other groups of related objects might need to be placed together spatially on the screen to emphasize their relationship.

In a static scene, observed from a fixed viewing specification, view-management decisions might be made in advance and used several times throughout the duration of an application. It is also common in both 2D and 3D interactive UIs to control some view management manually when possible. For example, a fixed area of the

screen may be dedicated to a menu, or the user may explicitly control the positions and sizes of permanent menus, temporary pop-up menus, windows or other widgets. However, hard-wired or direct-manipulation control is problematic when applied to dynamic scenes that include autonomous objects, and to scenes viewed using a head-tracked display: frequent and unpredictable changes in object geometry or viewing specification result in continual changes in the spatial and visibility relationships among the projections on the view plane. This often results in unwanted occlusion of important objects, misplaced or overlapping objects that would give the user an undesired effect. For these cases, we use view-management decisions made programmatically at real time so that they can take dynamic changes into account to prevent unwanted layouts.

In augmented reality systems, the system annotates the surrounding world with virtual objects to assist the user in understanding their environment. This is particularly challenging since the system may have no way to control the behavior of the physical objects; it is left with controlling only the overlaid information. The system needs to limit the virtual information shown so that the user is not obstructed from viewing and interacting with the real world environment. The system also needs to show enough information to be helpful. View management constraints are especially useful in these situations by guaranteeing visibility of important physical objects while placing labels and other annotations near the projections of the objects to which they refer.

Our system can also assist users in augmented reality using other methods. We describe a situation-awareness aid, based on a world in miniature (WIM) (Stoakley, Conway, & Pausch, 1995) or exocentric “god’s eye view” (Furness, 1986), that is intended to provide 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 to the user within that environment. To manage viewing the situation-awareness aid, the user’s head pitch controls the aid’s position, scale, and orientation. This allows some user initiated view management control: by using head movement, users can easily change their focus of attention between the real world and the virtual WIM. We emphasize that the View Management component is not only a system initiated automatic mechanism, but it takes into account both the user and system functioning together to manage what is seen.

2. EXPLORING VIEW MANAGEMENT

A prototype view-management module has been developed to explore some important features we envision it might contain. We focus on a real-time, interactive interface module that provides mechanisms for specifying constraints on the projections of objects on the view plane. It also provides the ability to add interface components that can interact with these specified 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 translation, scale, transparency, and inter-object spatial relationships that should be maintained.



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

To solve the constraints based on the visibility of 3D objects, we use interactive visible-surface determination algorithms to compute an efficient approximation of the objects' projections, based on upright 2D rectangular extents. For computing placements that avoid occlusion, we use the same algorithms to compute the screen space where objects' projections do not exist (i.e., the empty space). We use these representations to develop a variety of view-management strategies, including ones that take into account decisions made during previous frames to minimize frame-to-frame visual discontinuities.

Figure 1 demonstrates an example of how an augmented reality application uses view management to solve visual constraints. The scene is photographed through a see-through head-worn display from the perspective of one user in a collaborative augmented reality environment (Arthur et al., 1998; Billingham, Weghorst, & Furness, 1998; Butz, Höllner, Feiner, MacIntyre, & Beshers, 1999; Szalavari, Schmalstieg, Fuhrmann, & Gervautz, 1998). Two users are sitting across from each other, discussing a virtual campus model located between them. In response to a request from the user whose view is displayed, all virtual campus buildings have been labeled with their names. Each label is scaled within a user-selectable range and positioned automatically. A label is placed either directly within a visible portion of its building's projection. If there is no space to accommodate a legible label, it is placed near the projection of the building, but not overlapping other



Figure 2. Screen shot of a 3D interactive campus application

buildings or annotations, and is connected to its building by an arrow. Additional annotations include a meeting agenda (shown on the left), and popup information related to a building (in this case, shown on the right). The projections of annotations avoid other objects, including the visible user's head, to allow a direct line of sight between users. The popup information consists of a virtual copy of a building for examination, along with related information aside it. The virtual copy of the building is screen-stabilized in the spirit of the "Fix and Float" object movement technique (Robertson & Card, 1997), but is constrained such that it continually avoids occluding other objects as well.

Figure 2 shows an interactive 3D virtual application that allows users to explore information about buildings on our campus. Visible buildings are initially labeled to show identity, and to convey what is selectable. When a user selects a building, more information is shown and placed close to the building's visual projection with a leader line back to the building to convey the information's relationship. A virtual avatar is shown in the model with a text balloon, similar to comic balloons, that is constrained at a fixed position relative to the projection of the avatar's mouth. All of these annotations are laid out so they do not overlap each other or any of the buildings projections. The careful placement of these annotations allows the view of the 3D scene to continue, while still providing the user with useful information.

All annotations in these examples are placed automatically by our view-management module, personalized for each individual view. The personalized

annotations appear as local variations to each view in the environment, similar to what has been used in other distributed graphics packages (Hesina, Schmalstieg, Fuhrmann, & Purgathofer, 2000; MacIntyre & Feiner, 1998).

3. RELATED WORK

Some of the earliest work on automated view management in user interfaces was performed in the design of tiled window managers that automate the placement of non-overlapping windows (Teitelman, 1984), and of constraint-based window managers for both tiled (Cohen, Smith, & Iverson, 1986) and non-tiled windows (Badros, Nichols, & Borning, 2000) that enforce spatial constraints among objects. Although these systems managed multiple components or applications, none of them address these relationships in 3D environments.

Researchers have often labeled and annotated objects in virtual and augmented environments without taking into account visibility constraints (Fairchild, Poltrock, & Furnas, 1988; S. Feiner, MacIntyre, Haupt, & Solomon, 1993; Starner et al., 1997; Sutherland, 1968) (Mori, Koiso, & Tanaka, 1999). In contrast, graph layout algorithms (Battista, Eades, Tamassia, & Tollis, 1999) and label placement algorithms for point, line, and area features on 2D maps and graphs (Christensen, Marks, & Shieber, 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 et al., 1995) 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 algorithm 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. (Ahlberg, Williamson, & Shneiderman, 1992) show how direct manipulation and user feedback provide a useful way for users to filter information from a database. In virtual and augmented reality, there have been filtering based on user's objectives and proximity to other objects in the 3D environment (Julier et al., 2000). These techniques provide useful results, but lack the ability to make filtering decisions based on visibility or occlusion. There are many applications in which multiple objects may need to be displayed such that they are located near objects to which they are related or avoid occluding or being occluded by other objects.

A number of researchers have developed approaches for automatically avoiding 3D occlusion of selected objects. Some of these rely on controlling the viewing specification (He, Cohen, & Salesin, 1996; Phillips, Badler, & Granieri, 1992), which is not possible in head-tracked environments in which the viewing specification is slaved to the user's head. Others utilize illustrative effects, such as

cut-away views and transparency (S. Feiner & Seligmann, 1992) or line style (Steven Feiner, MacIntyre, & Seligmann, 1993; Kamada & Kawai, 1987), without altering object geometry. Visual access distortion (Carpendale, Cowperthwaite, & Fracchia, 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 that desired occlusion relationships with other objects are maintained.

4. OBJECT PROPERTIES AND CONSTRAINTS

Each of the objects in our scene 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 interface). This means that there can be multiple sources of change, and multiple 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 layout (currently visibility, position, size, and priority) and transparency of objects in a 3D scene. These constraints can be set explicitly by interface components, by the user, or they can be defined to change based on other properties or relationships. Some constraints might need significant processing to satisfy. Although the view-management component manages how much processing it uses to some extent, it is graceful to precisely specify what layout management constraints are needed throughout the application.

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 paper include:

- Pavement and grass patches in the campus scene can always be occluded by any 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, a "speech balloon" may be constrained to be above and near an area feature (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 eyepoint. For stereo viewing, this distance will control the distance users' eyes 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.

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

Each object can be associated with a priority. 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.

5. AUTOMATIC SYSTEM DRIVEN VIEW MANAGEMENT

Computing visible 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 in the application for each frame rendered. However, since some of these computations might take a considerable amount of time to compute, asynchronous execution of the pipeline for real-time applications, as described in the Decoupled Simulation Model (Shaw, Liang, Green, & Sun, 1992), would preserve interactive rendering frame rates but would sacrifice 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 these computations in our implementation.

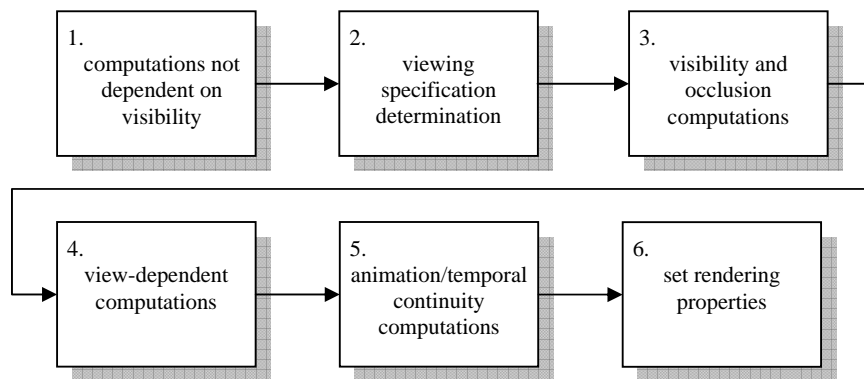


Figure 3. View Management Pipeline

5.1. 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, render graphics dependent on input from tracker hardware at real time. 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., in determining viewing specification, step 2) and setting the rendering 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 structure initializations can happen in this period when the system knows data structures are needed further down the pipeline.

5.2. Viewing Specification Determination

For each 3D object in the scene, we need to determine projection parameters that include eye point (two for a stereo view), orientation, and other viewing parameters (i.e., field of view, clipping planes, etc.). 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 where the view follows a predetermined path. Real-time applications determine the user's eye point and orientation by using input from tracker hardware, along with a set of transformations based on where the physical tracker exists and how the display is worn by the user. Other real time 3D applications can be controlled using direct manipulation using other devices, such as a mouse. Pre-determined applications, such as videos and animation, can be manually planned or can also have some automated assistance (He et al., 1996; Phillips et al., 1992). These computations might also include screen space computations to automatically choose camera placement. The situation awareness aid, described in section 6, presents a viewing specification for a world in miniature that is dependent on the user's head pitch, which is computed from the viewing specification for the physical world. In this AR example, two viewing specifications, for the physical world and the WIM, are computed from tracker input for rendering.

5.3. Visibility and Occlusion Computations

After determining the viewing specification, the constrained objects' visibility, occlusion and depth values are computed to help satisfy the constraints defined in Section 4. In order to satisfy these complex constraints, we use a 2D space representation (Bell & Feiner, 2000) to represent the visible spaces of the projections of 3D objects on the view plane. This representation of an arbitrary, possibly multifaceted spatial region consists of a list of rectangles that are axis-aligned, possibly overlapping, and considered to be the largest rectangles that lie

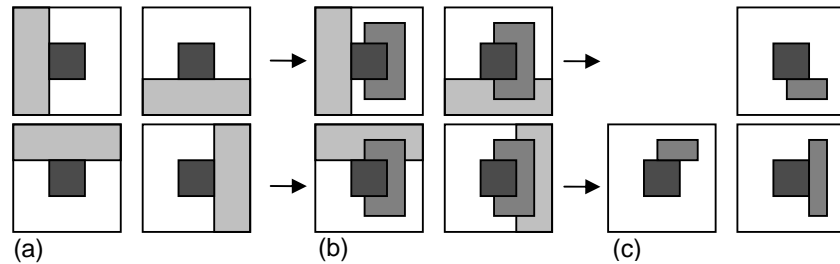


Figure 4. Adding a new object to the view-plane representation. (a) original object (dark grey), shown separately with each of the four original largest empty spaces (light grey) (b) New object extent (medium grey) is added. (c) Resulting visible largest spaces of new object extent (medium grey)

inside that region. For example, in figure 4(a), the four rectangles are considered to be the largest empty space rectangles for the region inside the rectangular space that does not include the dark grey rectangle. None of these largest space rectangles can get any larger and still remain completely inside the represented region.

By searching through these rectangles, it is possible to find placements inside the region that match the specified constraints (Section 5.4). For satisfying some constraints, such as to avoid occlusion, we can use the empty-space representation (Bell & Feiner, 2000) to compute the region of the screen where a set of objects do not exist. For other constraints we might need to compute the visible space of one object, or a combination of multiple objects. There are two visible-surface-determination algorithms we have used to generate these representations: one using a Binary Space Partition Tree (BSP) algorithm (Thibault & Naylor, 1987), and one using a hardware-accelerated z-buffer.

5.3.1. Visibility Determination using a Binary Space Partition (BSP) Tree

We can use the Binary Space Partitioning (BSP) Tree algorithm to efficiently determine the visibility order 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 objects' 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 viewing specification to find the visible portions of the scene's objects.

We use the 2D space representation, described in Section 5.3, to determine approximations of the full and empty portions of the view plane. The approach we introduce here provides an efficient way to compute the visible area of each object's

projection, while the resulting largest empty-space representation is used to satisfy constraints that avoid all other objects' projections.

As shown in Figure 4, 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 performed 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 (Figure 4b).

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 split prior to BSP tree construction to better approximate a large object by the upright extents of a set of smaller objects.) Therefore, a node's visible rectangles must be coalesced with those of all previously processed nodes from the same object to create the list of largest rectangles in the union of the nodes. Details of the coalescing algorithm is further described by (Bell, Feiner, & Höllerer, 2001).

5.3.2. Visibility Determination using a Z-buffer

We can also compute the 2D spatial representation of visible regions of objects in a 3D scene using a z-buffer algorithm, first developed by (Catmull, 1974). By exchanging objects' appearances with distinct colors to render another image with the same viewing specification, we are able to use the frame buffer as an *object buffer* (Atherton, 1981), where each pixel value defines which object is visible. Since antialiasing does not occur when rendering polygons in this image, we might run into similar problems in shading (Foley, Dam, Feiner, & Hughes, 1996), when two adjacent polygons of the same object fail to share a vertex on an adjacent edge. In this case, it is possible that all pixels within the projected area of this edge will not be rendered. To avoid this, we add extra vertices to any polygons in our model that might have this problem.

We use a sweep algorithm to convert the resulting pixel representation to the spatial representation needed for satisfying 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 wide dimension of the image keeping track of the last position 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 height 1) and combined into the current object's visible space representation. The current object and last position are then reset, and the algorithm continues. Since only rectangles that can be adjacent to a rectangle on the current line is one combined from the previous line, a 2D interval tree of all resulting rectangles from the previous line is kept for a 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 largest visible space rectangles for each visible region consist of all resulting rectangles including the rectangles that have been combined on the last scan line.

5.3.3. Performance and Accuracy

Since most current graphics hardware implements the z-buffer algorithm, the amount of time to compute the pixel representation is low. However, the conversion to our representation needs a lot of computation time, but it is more dependent on the number of pixels than the number of objects since the hardware implementations do quite well rendering a large number of polygons.

Rendering the z-buffer at a lower resolution than the graphics accelerates the algorithm using less pixels sacrificing accuracy. In this case, each pixel in the visibility algorithm represents a rectangular region in the final image in which the object might not have complete visibility.

Using the BSP tree, accuracy can also be traded for performance. Objects are approximated by the upright rectangular extents of their BSP tree nodes, and can have better approximations if more nodes are generated. Since there are more projections, more processing is involved.

5.4. View-Dependent Computations

The resulting visibility representations, consisting of the largest visible rectangles of single objects, multiple objects, and empty spaces (i.e., background), can support a variety of layout strategies. Queries on these visible regions are designed for choosing appropriate positions and sizes for new objects within these areas. For placing text, this is equivalent to area feature labeling (Roessel, 1989) in a 3D

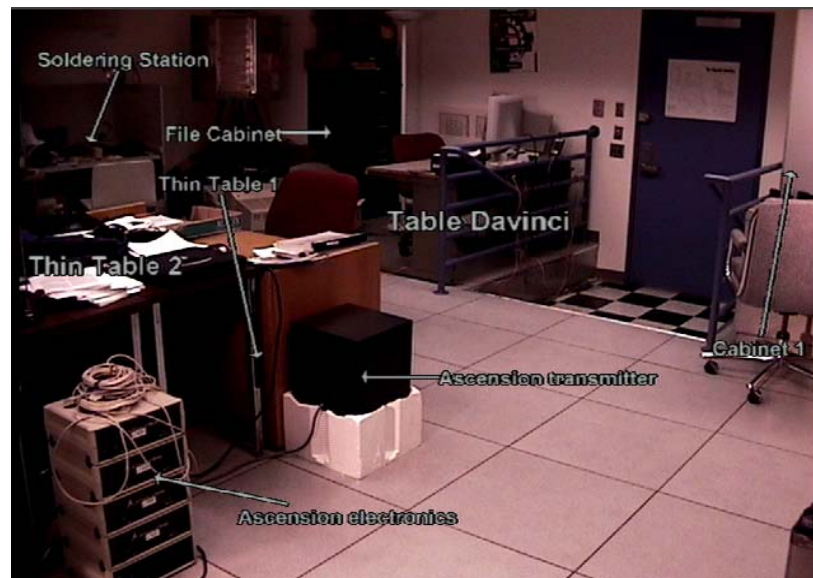


Figure 5. Browsing an augmented reality environment

environment where 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 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 queries done in (Bell & Feiner, 2000). We describe 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 are used to either directly place objects or as input to the animation computations (Section 5.5) to produce the final placement.

There are many cases in 3D interfaces when the application wants to point something out the user. Figure 5 shows examples of a browse mode, where the system tries to label as many objects as it can, making sure that the labels are not ambiguous. In this case, we use a two-tiered approach that creates both internal and external labels, ensuring that no label is occluded by any object (including other labels) and no label occludes any part of an object except the object it labels.

5.4.1. Internal Label Placement

For each object to be labeled we determine the internal largest visible rectangle 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 is blocked by the label if the object has a high priority. For efficiency, we approximate the dynamic size of a 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. However, for rendering high-quality hinted fonts without negative antialiasing effects, lookup tables can be used to keep the sizes of the labels at each font size to determine which size fits best. We have only used this when pre-processing a sequence of images because it requires much more processing time.

5.4.2. 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 region (i.e., the visible region no objects are projected) for a rectangle that can contain the label within a user-settable size range, and within a user-settable distance from the object. If the label is allocated within this space, it will neither be occluded by nor occlude any other object. If no such space can be found, then no label is 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 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.

5.4.3. Other Placements

Applications also might want to show more information about objects in the scene (Figures 1,2,6). Some information is placed similar to how labels are placed externally to related objects, possibly with different size and aspect ratio constraints. Some information, like 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, as in 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.



Figure 6: Image taken directly through an optical see-through head-worn display of restaurant and related popup information

5.5. 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. From informal feedback, when the viewing specification is controlled by a mouse, animation is not preferred. This is because once the user stops controlling the view, the scene stops, but the annotations keep moving. However, in an HMD application, annotations jitter and jump around so that it is difficult to read them.

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 describe a technique that is similar to traditional keyframe animation (Lasseter, 1987): comparisons between the results of the discrete queries (discussed in 5.4) and additional queries specific for animation (section 5.5.1) automatically determine the keyframes at real time for determining placement destinations, while interpolation computes the animation between the keyframe placements, or the “inbetweening”. Instead of introducing movement, this technique prefers that the annotations are continuously rendered at the keyframes for positional stability, while inbetween frames are rendered minimally: only when movement between keyframes is necessary.

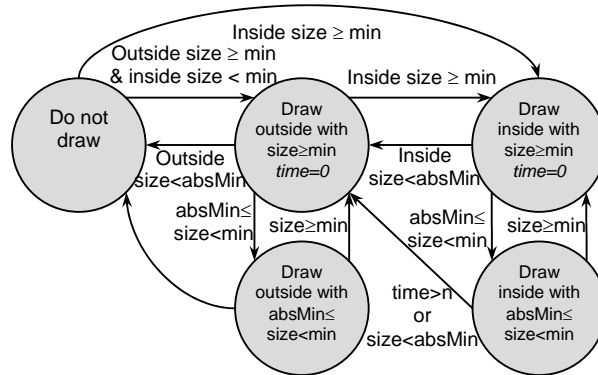


Figure 7: State hysteresis in area feature labeling. Area features are preferentially labeled inside rather than outside. Labels are promoted from left to right through comparison with min and demoted from right to left through comparison with $absMin$.

5.5.1. 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 external one, or from being displayed to not being displayed. Some UIs use three constraints on the size of an object being displayed: minimum size, maximum size, and preferred size (e.g., Java 2D). 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.

5.5.2. 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, 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 section 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 referred object could have moved because of view specification change (i.e., user's head movement), the last placement is not valid for use in finding the closest layout. Instead, we compute the change relative to the projection of the referred 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 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. If they are the same we use it; 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.

5.5.3. Interpolation

Once a keyframe is chosen, the current frame is chosen by interpolating the layout parameters from the previous frame to the keyframe. We use constant 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.

5.6. Set Rendering Properties

Once all of the view management pipeline computations are finished, the properties of the rendered objects need to get updated. Although the pipeline can run asynchronously with the rendering, as described in the beginning of Section 5, inconsistent results can occur if the rendering properties for an object do not get set before it's rendered. Therefore, all rendering properties need to get set before rendering begins, assuming that any object can effect another object's projection. Some of these dependencies could be eliminated, but usually not necessary since setting values takes minimal time to execute.

6. USER DRIVEN VIEW MANAGEMENT FOR AUGMENTED REALITY

In augmented reality, we are interested in assisting the user to interact with and obtain information about the physical environment. In our system, users can control labels, annotations, and other information by indirect manipulation; a mouse or other wireless device controls a menu to select filtering modes or manipulate objects' properties and constraints. Selecting a label reveals more information which can

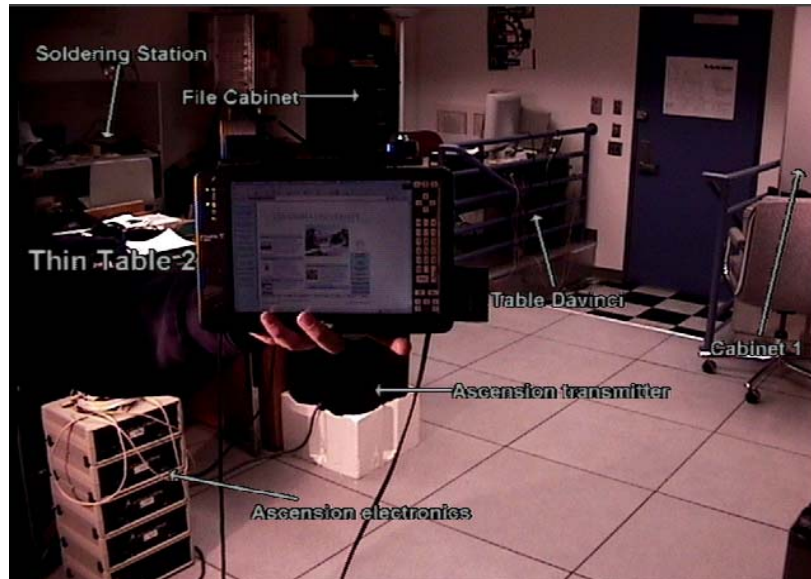


Figure 8: Tracked hand-held display takes priority over labels.

present buttons or other widgets to give even more options for getting additional information.

These interactions with virtual objects, although intuitive and useful, require almost the user's full attention to control and receive the information. We have explored some ways of using direct manipulation techniques to create interfaces that have less demand of attention. (Shneiderman, 1983) states the essential ingredients to direct manipulation include immediate response to actions, incremental changes, and reversible effects. These techniques, when applied to AR, can help the user balance their attention between the real world and virtual overlaid objects, and can interactively control what is seen virtually.

6.1. 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. Since the user wants to look at the screen of the tracked hand-held computer, it is defined as having a higher priority than any of the objects in Figure 5, and no associated label. As a result, much of "Table Davinci" is obscured by the hand-held computer, resulting in its label changing 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.

Instead of using a tracked hand-held computer, it is also possible to use a tracked hollow rectangular box that has the same shape. Instead of having the system showing labels for all objects seen, the box could act as a direct way for the user to ask the system to label any object projected within its projection.

6.2. Control of the Situated Awareness Aid

The situation awareness aid provides interactive control on a virtual 3D model of the environment, based on a "world in miniature", or WIM, by continuously responding to the user's head orientation. Using head pitch alone, the user can choose between focusing on the physical world with a small WIM view or exclusively on the WIM.

The WIM's yaw (y -axis orientation) is kept so that it is always aligned with the environment, as in (Darken & Sibert, 1993). Figure 9 shows screen-shots of how the control of the WIM works in our laboratory environment. In (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 (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 moves up slightly higher in the viewport. In (c), the user looks nearly straight down, and in response 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 WIM in our outdoor system that uses both texture-mapped aerial photography as well as 3D models of individual buildings.

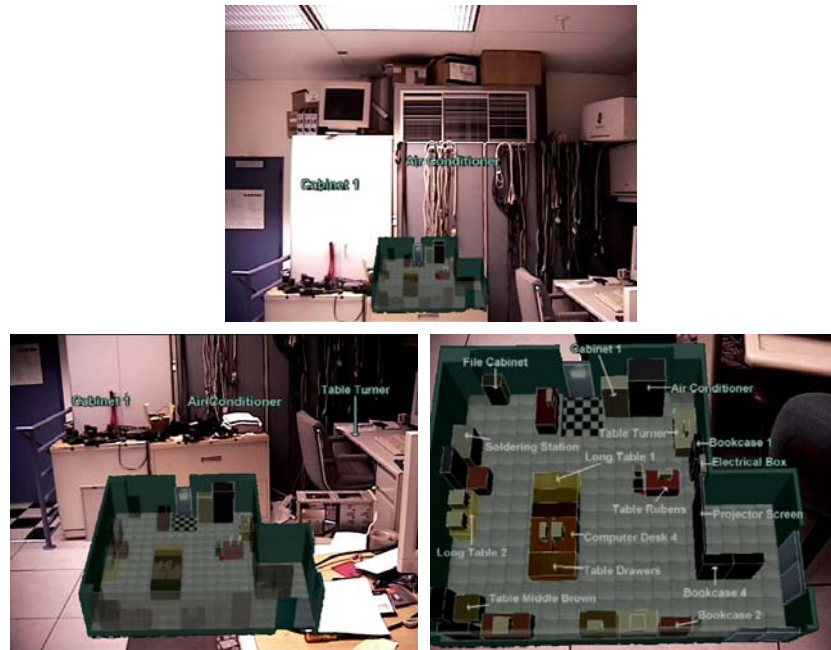


Figure 9. Annotated situation-awareness aid. (a) Looking slightly upward. (b) Looking slightly downward. (c) Looking nearly straight down.



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

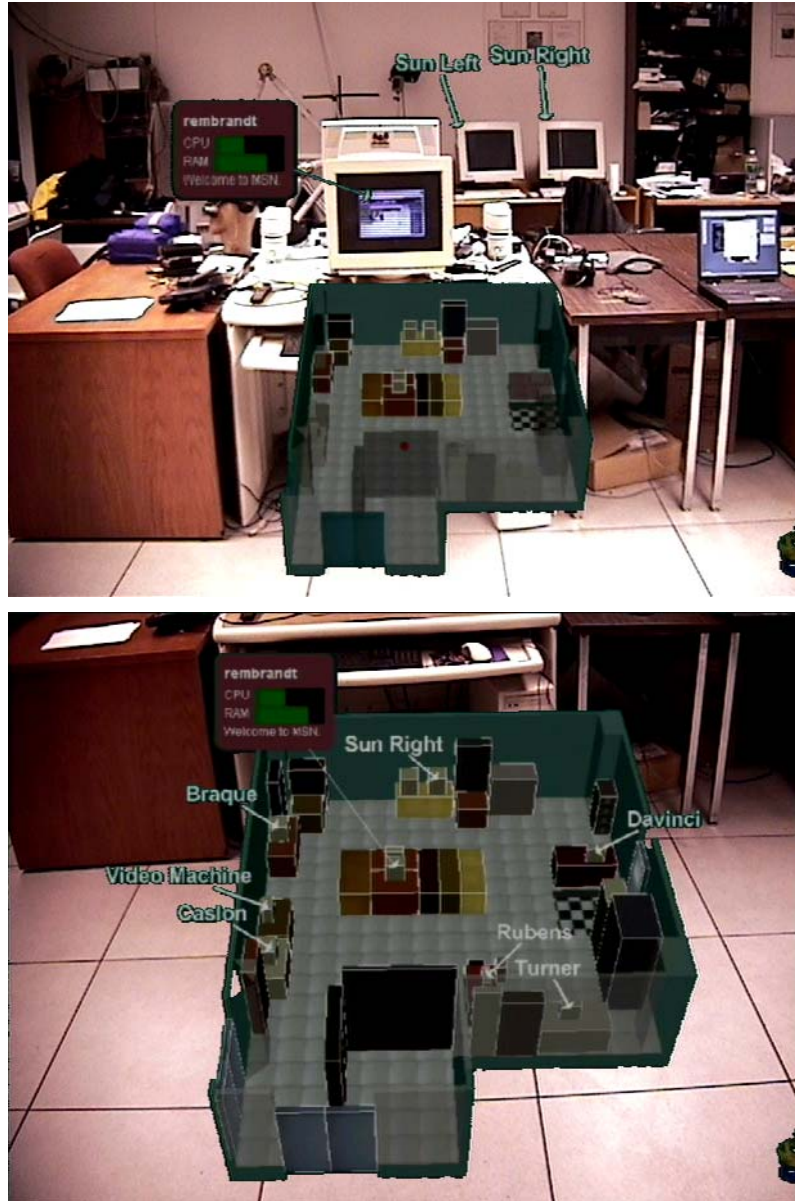


Figure 12. Transfer of annotations between the physical world and virtual aid. (a) Popup annotation provides information about the computer monitor to which its arrow points at the center. (b) As user looks down, annotations are enabled in the aid, and the popup's arrow now points at the monitor's representation in the aid.

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 WIM: 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. We linearly interpolate each dependent variable from min to max between the head pitch trigger values. More information on the exact implementation can be found in (Bell, Höllerer, & Feiner, 2002).

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

6.3. Annotation Assistance

The WIM provides us with another platform for assisting the user with more information and interaction. In figure 11(c), labels are placed relative to the projections of the virtual objects in the WIM 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 WIM.

Popup annotations behave in the exact same way. Figure 12(a) shows an annotation placed relative to the visual projection of a physical object while avoiding other important objects, including other physical objects, labels, and the WIM. As the user looks down at the WIM, 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 12(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. Also, information can be selected from the virtual objects in the WIM about physical objects that might not be currently visible in the real world from where the user is currently positioned.

7. CONCLUSIONS

We have described a framework for view management in virtual and augmented reality applications. While many of the underlying algorithms described may change with further development and refinement to increase performance, added constraints, and increased functionality, the basic concept of view management is fundamental to creating 3D interactive applications: placing visual constraints to

automatically control the visibility and spatial layout of objects allows manageability to what users see in a dynamic 3D environment. Our design is particularly beneficial since it only solves the constraints that are given, and does not impose overhead if none are specified.

Since constraints can be changed throughout the application, this allows an infinite number of possibilities for the application and how they are used. We are currently designing a knowledge-based infrastructure to allow us to flexibly control how constraints are used so that it is easier to handle more complex situations.

8. ACKNOWLEDGMENTS

This work was supported by ONR Contracts N00014-99-1-0249, N00014-99-1-0394, and N00014-99-1-0683, NSF Grant IIS-00-82961, NLM Contract R01 LM06593-01, and gifts from Intel, Microsoft, and Mitsubishi.

One of the authors was supported by an IBM Fellowship.

9. REFERENCES

- Ahlberg, C., Williamson, C., & Shneiderman, B. (1992). Dynamic Queries for Information Exploration: An Implementation and Evaluation. In *Proc. ACM CHI '92* (pp. 619--626): ACM press.
- 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. In *Proc. SIGGRAPH '81* (pp. 279-287): ACM Press.
- Badros, G. J., Nichols, J., & Borning, A. (2000). SCWM--An Intelligent Constraint-Enabled Window Manager. In *Proc. AAAI Spring Symposium on Smart Graphics*. Stanford, CA.
- 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). Dynamic Space Management for User Interfaces. In *Proc. UIST '00* (pp. 239--248). San Diego, CA: ACM Press.
- Bell, B., Feiner, S., & Höllerer, T. (2001). View Management for Virtual and Augmented Reality. In *Proc. UIST '01* (pp. 101--110). Orlando, FL: ACM Press.
- Bell, B., Höllerer, T., & Feiner, S. (2002). An Annotated Situation-Awareness Aid for Augmented Reality. In *Proc. UIST '02* (pp. 213--216). Paris, France: ACM Press.
- 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*. 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).
- Cohen, E. S., Smith, E. T., & Iverson, L. A. (1986). Constraint-Based Tiled Windows. *IEEE Computer Graphics and Applications*, 6(5), 35--45.
- Darken, R. P., & Sibert, J. L. (1993). A Toolset for Navigation in Virtual Environments. In *Proc. UIST '93* (pp. 157--166). New York, NY: ACM Press.

- Fairchild, K. M., Poltrock, S. E., & Furnas, G. W. (1988). SemNet: Three-Dimensional Graphic Representation of Large Knowledge Bases. In R. Guindon (Ed.), *Cognitive Science and its Applications for Human-Computer Interaction* (pp. 201--233). Hillsdale, New Jersey, U.S.A.: Lawrence Erlbaum Associates.
- Feiner, S., 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, S., MacIntyre, B., & Seligmann, D. e. e. (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., Dam, A. v., Feiner, S., & Hughes, J. (1996). *Computer Graphics: Principles and Practice, 2nd Ed. in C.* Reading, MA: Addison-Wesley.
- Furness, T. (1986). The Super Cockpit and Its Human Factors Challenges. In *Proc. Human Factors Society 30th Annual Meeting* (pp. 48--52). Santa Monica, CA.
- 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). Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics. In *Proc. VRST '99* (pp. 74--81). N.Y.: ACM Press.
- Imhof, E. (1975). Positioning names on maps. *The American Cartographer*, 2(2), 128--144.
- Julier, S., Lanzagorta, M., Baillet, 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. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4), 35--44.
- 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). Automatic Viewing Control for 3D Direct Manipulation. In *Proc. SIGGRAPH '92* (pp. 71--74). Cambridge, MA: ACM Press.
- Robertson, G., & Card, S. (1997). Fix and Float: Object Movement by Egocentric Navigation. In *Proc. UIST '97* (pp. 149--150). Banff, Alberta: ACM Press.
- Roessel, J. W. v. (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., Liang, J., Green, M., & Sun, Y. (1992). The Decoupled Simulation Model for Virtual Reality Systems. In *Proc. CHI '92* (pp. 321--328). Monterey, CA: ACM Press.
- Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer*, 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). Virtual Reality on a WIM: Interactive Worlds in Miniature. In *Proc. CHI '95* (pp. 265--272). Denver, CO: ACM Press.
- Sutherland, I. (1968). A Head-Mounted Three Dimensional Display. In *Proc. FJCC 1968* (pp. 757--764). Washington, DC: Thompson Books.
- Szalavari, Z., Schmalstieg, D., Fuhrmann, A., & Gervautz, M. (1998). Studierstube: An Environment for Collaboration in Augmented Reality. *Virtual Reality*, 3(1), 37--48.
- Teitelman, W. (1984). A Tour Through CEDAR. *IEEE Software*, 1(2), 44--73.
- Thibault, W., C., & Naylor, B., F. (1987). Set operations on polyhedra using binary space partitioning trees. In *Proc. SIGGRAPH '87* (pp. 153--162): ACM Press.