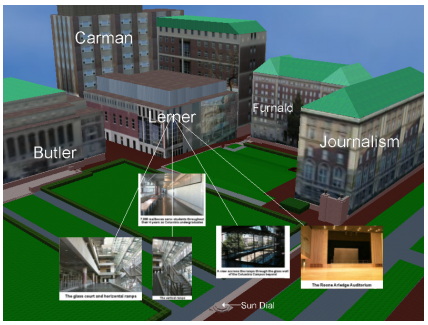
UIST 2001 (ACM Symp. on User Interface Software and Technology), Orlando, FL, November 11-14, 2001, pp. 101-110

View Management for Virtual and Augmented Reality

Blaine Bell *Steven Feiner* *Tobias Höllerer*
Department of Computer Science
500 W 120th St., 450 CS Building
Columbia University
New York, NY 10027
{bell,feiner,htobias}@cs.columbia.edu



ABSTRACT

We describe a view-management component for interactive 3D user interfaces. By *view management*, we mean maintaining visual constraints on the projections of objects on the view plane, such as locating related objects near each other, or preventing objects from occluding each other. Our view-management component accomplishes this by modifying selected object properties, including position, size, and transparency, which are tagged to indicate their constraints. For example, some objects may have geometric properties that are determined entirely by a physical simulation and which cannot be modified, while other objects may be annotations whose position and size are flexible.

We introduce algorithms that use upright rectangular extents to represent on the view plane a dynamic and efficient approximation of the occupied space containing the projections of visible portions of 3D objects, as well as the unoccupied space in which objects can be placed to avoid occlusion. Layout decisions from previous frames are taken into account to reduce visual discontinuities. We present augmented reality and virtual reality examples to which we have applied our approach, including a dynamically labeled and annotated environment.

CR Categories and Subject Descriptors: H.5.1 [**Information Interfaces and Presentation**] Multimedia Information Interfaces—*Artificial, augmented, and virtual realities*; H.5.2 [**Information Interfaces and Presentation**] User Interfaces—*Graphical User Interfaces, Screen design*; I.3.6 [**Computer Graphics**] Methodology and Techniques—*Interaction Techniques*; I.3.7 [**Computer Graphics**] Three-Dimensional Graphics and Realism—*Virtual Reality*.

Additional Keywords and Phrases: view management, environment management, annotation, labeling, wearable computing, augmented reality, virtual environments

1. INTRODUCTION

Designing a 3D graphical user interface (UI) requires creating a set of objects and their properties, arranging them in a scene, setting a viewing specification, determining lighting and rendering parameters, and deciding how to update these decisions for each frame. Some of these decisions may be fully constrained; for example, a simulation may determine the position and shape of certain objects, or the viewing specification may be explicitly controlled by the user. In contrast, other decisions must be resolved by the UI designer. We are especially interested in

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

decisions that determine the spatial layout of the projections of objects on the view plane. We refer to these decisions as *view management.* For example, some objects may be sufficiently important to the user's task that they should not be occluded, while the members of a group of related objects may need to be placed together to emphasize their relationship.

In a static scene, observed from a fixed viewing specification, view-management decisions might be made in advance, by hand, and hold throughout the life of an application. It is also common in both 2D and 3D interactive UIs to control 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 of permanent menus or temporary pop-up menus, or the positions and sizes of windows or widgets. However, hard-wired or direct-manipulation control is problematic when applied to dynamic scenes that include autonomous objects, and to head-tracked displays: continual 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. In these cases, view-management decisions must be made on the fly if they are to take dynamic changes into account.

Augmented reality applications are especially challenging in this regard. Virtual and physical objects reside in the same 3D space, and we may have no way to control the behavior of many of the physical objects. For example, the view through an optical see-through head-worn display includes all the physical objects that occupy the user's field of view in addition to the virtual objects, and the portion of the field of view that can be augmented may be relatively small.

In the real world, we make simple view management decisions routinely, and often subconsciously. For example, we push aside a centerpiece to enable eye contact with others at dinner, or we hold a city guidebook so that we can read a relevant section, while simultaneously viewing an historic building that it describes as we pass by on a bus. We are interested in automating view-management decisions in virtual and augmented reality, including ones that may be far too complex to tackle by hand in the real world. For example, to document an unfamiliar environment, we may wish to have labels or other annotations placed in or near the projections of the objects to which they refer, yet avoid blocking the user's view of other objects. In the real world, we may view a static annotated map, holding it so as to complement our view of the surrounding environment. Instead, we would like to have virtual annotations interspersed among the objects they describe, and reconfigured automatically and understandably to take into account changes in the objects themselves and how they are viewed.

1.1 View Management Testbed

To explore the potential for automated view management, we have developed a prototype view-management component that maintains relationships among the projections of virtual and real 3D objects on the view plane. The input to our view-management component includes objects that are tagged to indicate constraints on their properties, such as translation, scale, and transparency, and inter-object spatial relationships that should be maintained. The view-management component uses the upright 2D rectangular extents of the objects' projections in an interactive analytic visible-surface determination algorithm to create a dynamic and efficient approximation of the space occupied by the visible parts of the projections, along with the empty space in which other objects can be placed to avoid occlusion. We use this representation 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 shows examples of the kinds of visual constraints our prototype view manager interactively resolves. The scene is photographed through a see-through head-worn display from the perspective of one user in a collaborative augmented reality environment [34, 7, 6, 2]. 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, or, if the visible parts of the projection are deemed too small to accommodate a legible label, the label is placed near the building, but not overlapping other buildings or

annotations, and is connected to its building by an arrow. Additional annotations include an agenda, at the left, and images and text related to the buildings to which they are attached, which are invoked or dismissed by selecting their building.

The projections of annotations avoid other objects, including the visible user's head, to allow a direct line of sight between users. The user whose view is being displayed has created a copy of one of the buildings to examine more closely at the upper right. (The copy is a "world in miniature" [32], screen-stabilized in the spirit of the "Fix and Float" object movement technique [29], but constrained such that it continually avoids occluding other objects.)

All annotations are placed automatically by our view-management component, personalized for the individual view that a meeting participant has onto the otherwise shared graphical scene. The personalized annotations appear as local variations in a shared augmented reality environment, supported by a distributed graphics package similar in spirit to that of [25, 20].

In the remainder of this paper, we first describe related work in Section 2. Next, we examine the object properties and constraints we support in Section 3. We follow this by introducing the algorithms that we use to perform view management in Section 3.5, and our implementation in Section 5. Finally, we present our conclusions and a discussion of future work in Section 6.

2. RELATED WORK

Some of the earliest work on automated view management in 2D UIs was performed in the design of tiled window managers [35] that automate the placement of non-overlapping windows, and of constraint-based window managers for both tiled [11] and non-tiled [3] windows that enforce spatial constraints among objects. However, none of these systems addresses relationships in 3D environments.

Researchers have often labeled and annotated objects in virtual and augmented environments without taking into account visibility constraints (e.g., [33, 13, 14, 31, 27, 38]). In contrast, graph layout algorithms [4] and label placement algorithms for point, line, and area features on maps and graphs [9] 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 [9]. As described in Section 3.5, our work uses a greedy, non-optimal algorithm 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 a desired spatial relationship with selected objects.

In virtual and augmented reality, it is possible to avoid some view-management decisions by caching virtual objects out of the user's field of view (e.g., in a virtual tool belt or just above the view volume), and displaying them briefly when needed [26]. Filtering approaches, in general, can be used to limit the number of virtual objects being displayed [1, 23]. Nevertheless, 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 (e.g., [19, 28]), 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 [16] or line style [24, 15], without altering object geometry. Visual access distortion [8] 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.

3. OBJECT PROPERTIES AND CONSTRAINTS

Each of the objects in our scene has properties, some of which may be tagged as *controllable* or *constrained* by the user, the view-management component, or other components (e.g., a tracker or a simulation). The view-management component makes use of controllable properties to help maintain its constraints. We support a set of object properties that constrain the layout (currently visibility, position, size, and priority) and transparency of objects in a 3D scene. In our current testbed, constraints and properties can be set explicitly by the user. We are also exploring the use of rule-based approaches to set properties and impose constraints in response to changes in context and user interaction.

3.1 Visibility

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.
- Campus buildings can be occluded by certain other objects at certain times. Buildings 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.

3.2 Position

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).

3.3 Size

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.

3.4 Transparency

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

3.5 Priority

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.

4. VIEW-MANAGEMENT APPROACH

Our view-management component keeps track of the projections on the view plane of the visible portions of objects, using a representation that makes it easy to query where objects have been rendered, as well as where objects have not been rendered.

To ensure that constraints are satisfied in a dynamic environment, our system adjusts the position and size of the controllable objects at every frame. As shown in Figure 2, we first determine the view-plane representation for all non-controllable objects that are needed to satisfy the constraints. Then, we process each controllable object in priority order, determining its position and size. Each controllable object whose visibility constraints involve other controllable objects that have not yet been processed must also be added to the view-plane representation.

4.1 View-Plane Representation

For the sake of efficiency, we approximate each object by the upright rectangular extent of its projection on the view plane. (Note that this approximation is used for view-management only, not for rendering.) Our algorithm extends the 2D space management approach of Bell and Feiner [5]. The original algorithm is given an incrementally specified input set of possibly overlapping, axis-aligned, rectangles. It automatically maintains an efficient representation of the dual of this set of rectangles: the area that has not been rendered. Any object whose projection lies wholly within this unrendered area will not occlude or be occluded by any of the objects in the input. The representation of the unrendered area is especially easy to query because it consists of a set of axis-aligned "largest empty-space rectangles." Each largest empty-space rectangle has a height and width that is as large as it can be without overlapping any of the input rectangles. Thus, each of the largest empty-space rectangles is bounded on the left, right, top, and bottom by an edge of an input rectangle or an edge of the viewport.

Using this 2D representation in its original form, we can represent the projections of 3D objects as the upright rectangular extents of their projections on the view plane. This makes possible the following simple, but limited, technique, which we use as a starting point for the extensions described in this paper.

4.1.1 Preventing an object from occluding or being occluded by all others

Suppose that the axis-aligned rectangular extents of the projections of all scene objects have been added to the representation, and that we are given the extent *A* of some new object's projection. To determine locations within which *A* will neither occlude nor be occluded by any other existing object, we can compare *A* with each largest empty-space rectangle to find all largest empty-space rectangles that wholly contain it. The set of largest empty-space

```
for each frame {
    compute view-plane representation for all
        non-controllable objects needed to satisfy constraints
        // See Section 4.2
    for each controllable object in priority order {
        determine object position and size, based on constraints
            and temporal continuity // See Sections 4.3-4.5
        if object has visibility constraints involving controllable
            objects that have not yet been processed then
                add object to view-plane representation
    }
}
```

Figure 2. View Management pseudocode.

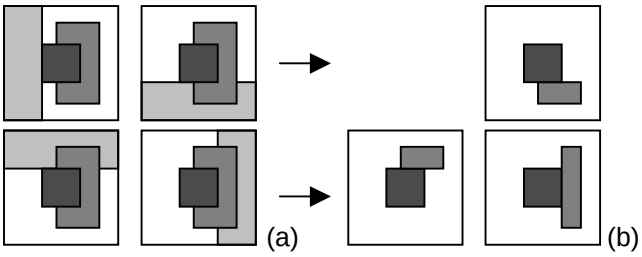


Figure 3. Adding a new object to the view-plane representation. (a) New object extent (medium grey) is added to original object (dark grey), shown separately with each of the four original largest empty spaces (light grey). (b) Resulting visible largest spaces of new object extent (medium grey).

rectangles that wholly contain *A* represents the full range of locations in which *A* can be placed to satisfy these constraints.

4.2 Visible-Surface Determination

The technique of Section 4.1.1 treats all 3D scene objects as the single undifferentiated 2D silhouette of the union of their projections. However, we need to determine the relationship of one or more 3D objects to the visible portions of one or more other 3D objects. To accomplish this, it is necessary to do visible-surface determination to determine which parts of the scene's objects are actually visible. Furthermore, we would like to represent visible surfaces such that new objects can be allocated within the projections of a desired subset of the existing objects (e.g., to place labels within or near the existing objects). In the remainder of this section and Sections 4.3–4.5, we discuss how we have extended the 2D space management approach to address these issues.

We perform visible-surface determination for view management by sorting the upright extents of the objects' projections in visibility order. (Visible-surface determination for interactive rendering is accomplished independently by the graphics hardware.) We use the Binary Space Partitioning (BSP) Tree algorithm [36] to efficiently produce 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 [36]. 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 [10].

4.2.1 Determining the visible portions of each object

For each frame, we traverse the BSP tree in front-to-back order relative to the eye to find the visible portions of the scene's objects. We extend the 2D space representation to determine approximations of the full and empty portions of the view plane. Much as our largest empty-space representation provides an efficient way to find suitable locations within the empty space, the approach we introduce here provides an efficient way to find suitable locations within the visible area of each object's projection.

As shown in Figure 3, 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 [30], 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 portion of the new node (Figure 3b).

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.

4.2.2 Coalescing lists of largest visible rectangles

To coalesce two lists of largest visible rectangles, we merge each visible rectangle from one list with the visible rectangles of the other list. This operation is similar to the deletion algorithm of [5], in which the largest empty-space rectangles that are revealed when a covering rectangle is deleted are combined with all adjacent largest empty-space rectangles. Unlike the deletion algorithm, we combine lists of largest visible rectangles incrementally, which means that some pairs of rectangles being processed may overlap. To handle these cases, the function that combines two

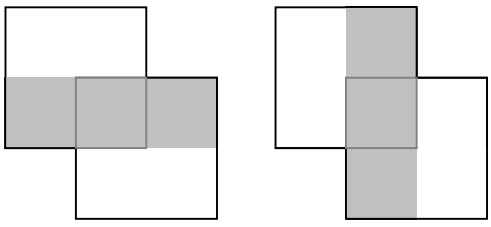


Figure 4. Two overlapping largest rectangles (white boxes) create an additional largest rectangle (grey boxes) in each dimension in which their combined extent is greater than that of either original rectangle.

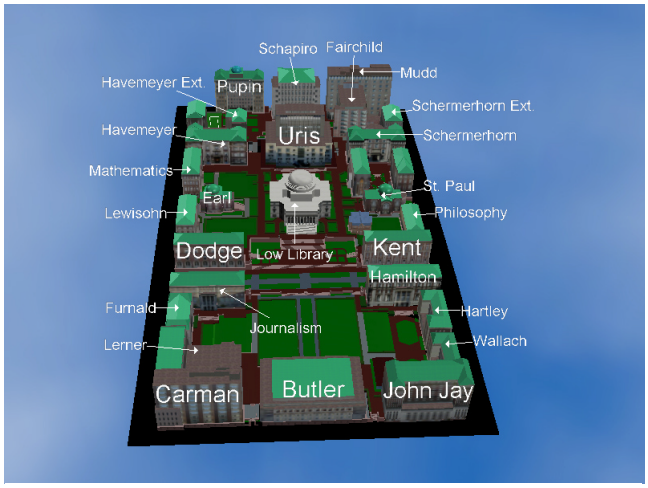


Figure 5. Long shot results in a majority of external labels.

Figure 6. Fewer external labels are used as the camera moves in.

largest visible rectangles must support overlap, creating up to two new largest visible rectangles, as shown in Figure 4, Thus for each rectangle in one list, we find all ways in which it can combine with the rectangles in the other list, and remove those that are enclosed by any other, to determine the largest visible rectangles in the union of the two lists.

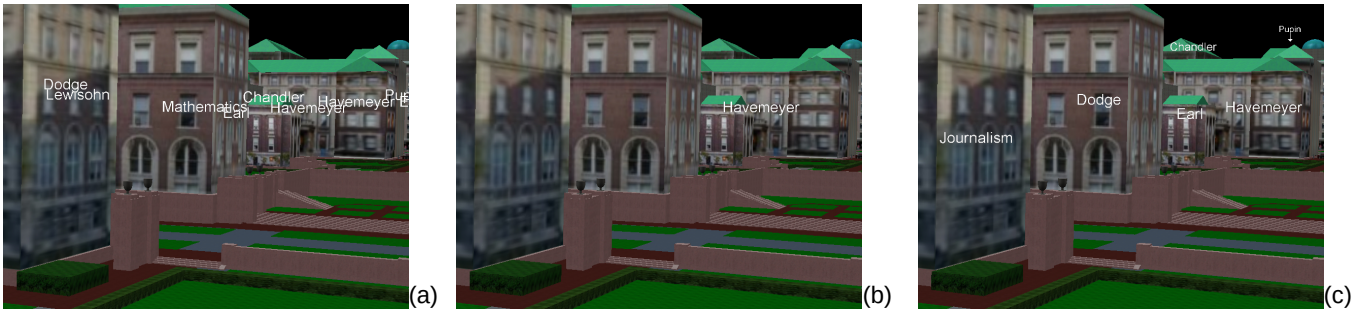
The resulting representation supports a variety of layout strategies. Each object is represented by the largest rectangles contained within its the visible parts of the object's projection. Thus, an appropriate position and size can be selected for a new object within any object's visible projection. This makes it possible to perform the equivalent of area feature labeling [37] in a 3D environment where objects can partially occlude each other; in this situation, we need to determine which parts of an object are visible, and select that one that would be best to contain a label.

When laying out a new object, a rectangle can be selected that lies within one of a set of objects. Alternatively, the coalescing algorithm can be used to merge the largest rectangles of multiple objects to create a meta-object that corresponds to the union of those objects. This enables the system to lay out a new object whose projection spans the space of multiple objects. Objects can either be coalesced as they are added to the representation or after the representation is built. If both the original objects and the coalesced objects are desired, the coalesced objects can be created in a second space manager representation.

Since full space objects are represented the same way as empty space, both can be coalesced; for example, to treat certain objects as empty space (e.g., the grass, pavement, and sky shown in the figures). Note that coalescing full space with empty space is *not* the equivalent of simply not using the full space objects when constructing the representation: some full space objects might obscure parts of others. For example, in our model the grass and pavement obscure underground infrastructure that would be treated as visible if the grass and pavement were not added.

4.3 Area Feature Labeling

To label area features, we adopt 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 any object except the object it labels. Figures 5-6 show examples of labels laid out by our system for the same scene imaged with different viewing



(a) (b) (c)

Figure 7. Area feature labeling comparison. (a) Naïve label placement at projected centroid of each building causes overlaps, and mislabeling because of buildings hidden partially or entirely. (b) Suppression of labels for buildings whose projected centroid is not visible. (c) Label placement using approach of Section 4.3.

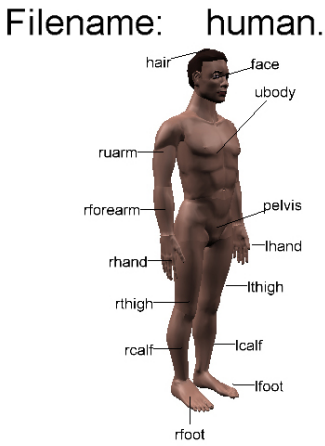


Figure 8. Labeling approach applied to named objects in VRML models. This variant uses only external labels.

specifications.

We first describe our approach without taking into account temporal continuity. For each object to be labeled, we determine the internal largest 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 copy is added to the scene and to the view-plane representation at the center of that rectangle to label the object internally

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 looks for a rectangle in the list of largest empty-space rectangles 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 [22]. 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. Labels can be created to lie in the view plane or, for stereo viewing, can be positioned to lie just on the viewer's side of the object being labeled (or just on the viewer's side of the object on which the external label is placed if it is closer).

Figure 7 compares our strategy with a simple layout approach that positions each label at the projected centroid of the object with which it is associated (7a), and a variant on this approach that suppresses labels whose object's centroid is not visible (7b). These other approaches result in overlaps and mislabelings because they do not take into account hidden parts of the buildings or conflicts in label placement. In contrast, each internal label in Figure 7(c) appears within the visible projection of its building and no labels overlap. Figure 8 shows a simple 3D modeling utility that uses our algorithm to label the named objects in

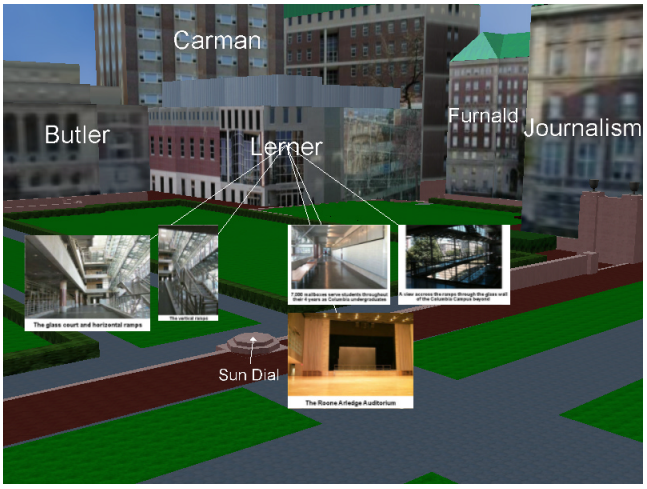


Figure 9. Pictorial annotations associated with building avoid occluding other objects, including Sun Dial.

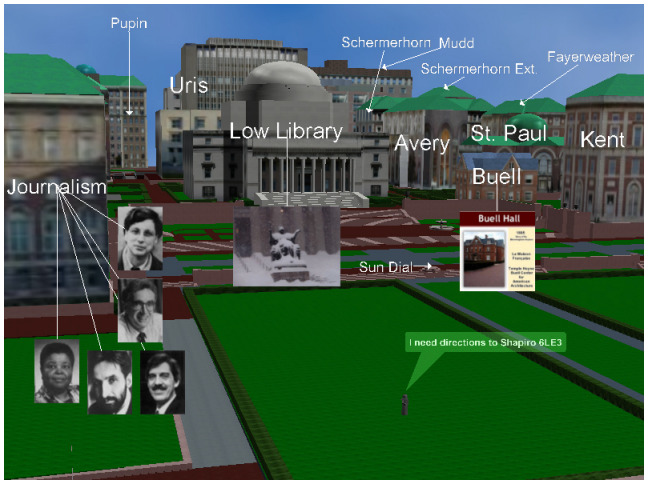


Figure 10. Annotations associated with multiple objects, including buildings and avatar for outdoor user. Building annotations were specified to have higher priority than avatar's annotation, causing avatar's annotation to avoid buildings and their annotations.

a VRML file, suppressing objects that are not visible. Because visibility is determined with approximations, some objects are incorrectly determined to be invisible (e.g., lforearm), an issue we address in Section 6.

4.4 Area Feature Annotations

In contrast to our area feature labeling approach, we also support area feature annotations that contain additional material that annotates an object, such as images and textual descriptions. Examples are shown in Figures 9–10. Each building annotation is connected to its object by a leader object to the center of the largest upright rectangle contained within the projection of the visible portion of the object. Each of these annotations is positioned within the smallest empty space rectangle that is closest to the object being annotated and which can contain the annotation. The text balloon attached to the outdoor user's avatar in Figure 9 is constrained relative to the avatar's mouth. Like labels, annotations avoid occluding each other as well as buildings.

4.5 Temporal Continuity

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 can be annoying. Therefore, we need to take into account decisions made during previous frames when laying out the current frame. We address this in three ways. First, we introduce hysteresis in the discrete state changes that can take place. Second, we try to make sure that objects being laid out occupy roughly the same position relative to their defining object (or to their screen position if screen-stabilized) as they did in the previous frame. Third, we interpolate between certain kinds of discrete changes.

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 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 11, 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

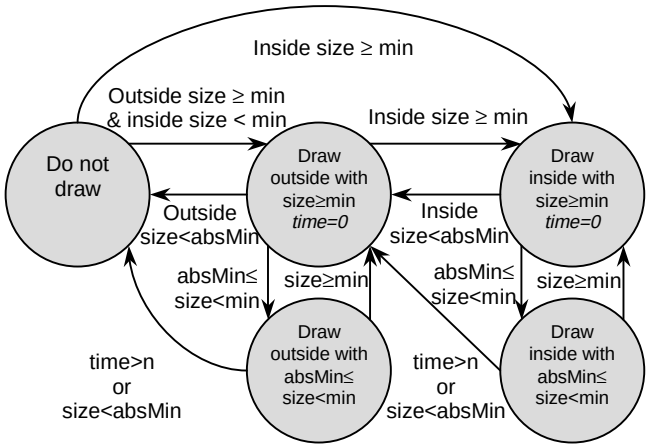


Figure 11. State hysteresis in area feature labeling. Area features are preferentially labeled inside rather than outside. Labels are promoted from left to right through comparison with min and demoted from right to left through comparison with absMin.

size no greater than its maximum size. The state diagram of Figure 11 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

For an object *L* being placed relative to object *A*, we compute two possible layouts: the best possible layout independent of the previous layout, and the closes t possible layout to the previous layout. For example, when *L* is an internal label for *A*, the best possible layout that we currently compute uses the visible space in *A* that can contain the largest allowable version of *L*. To determine the closest possible layout to the previous layout, we compute the position of *L*'s centroid in the previous frame relative to *A*'s unclipped width and height in the previous frame. We then use these proportions to compute from *A*'s unclipped width and height in the current frame, a predicted position L^C for *L*'s centroid. Next, we compare best and closest possible layouts. 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 and reset the timer.

4.5.3 Interpolation

To minimize the effect of discontinuous jumps during the state changes discussed above, *L* is interpolated from its previous position and scale to its new ones. In changing from internal to external labels, we also grow or shrink the arrow.

5. IMPLEMENTATION

Our view-management component has been implemented in Java 1.3 with Java 3D 1.2.1.01 [12]. The images in this paper were computed on a 1.4 GHz Intel Pentium 4

processor with 512MB RAM and a SONICBlue FireGL 2 graphics board, running Windows 2000. Note that the visible-surface processing performed by our algorithm is used for view management only; rendering is accomplished entirely through Java3D. While performance depends on the complexity of the scene, for the models used in this paper, our systems runs at about 10–25 frames per second in stereo for an 800x600 resolution Sony LDI-D100B head-worn display (with the view-plane representation computed for a single eye).

6. CONCLUSIONS AND FUTURE WORK

We have implemented a prototype view-management component for virtual and augmented reality, and have experimented with it in a set of applications that support dynamic labeling of area features in the environment, and the maintenance of visibility relationships among different objects. There are a number of issues that we will be exploring with our system:

Improved layout strategy. While our current layout algorithms run just fast enough for comfortable interaction, we believe that we could significantly improve their quality by incorporating additional constraints, especially on the placement of potentially ambiguous external labels. The map label placement literature includes a variety of heuristics that we do not currently use, including preferences for the direction in which labels should be placed relative to a feature, and the desirability of avoiding locations that are too near other objects [22]. Adding information to our view-plane representation about the objects that bound each largest empty-space would make it possible to evaluate potential placements more effectively. We intend to apply these techniques to pre-rendered material, in order to experiment with look-ahead and with algorithms that are more computationally intensive than is feasible in a real-time environment.

Layout quality depends in part on how well objects are approximated by their upright rectangular extents. This depends both on an object's intrinsic shape and on its orientation; a thin rectangular prism that lies along the view plane's diagonal is a particularly bad example. To address this, we intend to use hierarchical object representations that allow the system to choose an appropriate level of detail. For example, the rectangular prism might be expressed as a set of smaller prisms when their rectangular extents are determined to better approximate the object's projection.

Rule-based view management. While our current system uses constraints that are imposed explicitly by users, we are especially interested in exploring how knowledge-based component could control the view-management component in response to changes in the users' environments and tasks.

Usability studies. The algorithms that we have presented make it easy to design a wide range of different behaviors. To determine what will work best for users engaged in different tasks, we are beginning to design a series of usability studies. Our goal is to compare performance with and without different versions of the view-management component on modeling tasks in cluttered environments that would appear to be good candidates for its use.

ACKNOWLEDGMENTS

The research described here is funded in part 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. Ryuji Yamamoto created the campus model shown in the figures. Simon Lok, Andrew Cheung, William Chiong and Yoav Hirsch developed the software infrastructure that we used to create a GUI for specifying user parameters.

REFERENCES

[1] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proc. CHI '92*, pages 619-626, 1992.

[2] K. Arthur, T. Preston, R. Taylor II, F. Brooks, Jr., M. Whitton, and W. Wright. Designing and building the PIT: A head tracked stereo workspace for two users. In *Second Int. Immersive Projection Technology Workshop*, Ames, IA, May 11-12 1998. www.cs.unc.edu/Research/graphics/GRIP/PIT/doc/ipt-paper.pdf.

[3] G. J. Badros, J. Nichols, and A. Borning. SCWM-an intelligent constraint-enabled window manager. In *Proc. AAAI Spring Symposium on Smart Graphics*, Cambridge, MA, Mar.20-22 2000. (http://scwm.mit.edu).

[4] G. D. Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, USA, 1999.

[5] B. Bell and S. Feiner. Dynamic space management for user interfaces. In *Proc. ACM UIST 2000 (Symp. on User Interface Software and Technology), CHI Letters, vol. 2, no. 2*, pages 239-248, San Diego, CA, November 5-8 2000.

[6] M. Billinghurst, S. Weghorst, and T. Furness III. Shared space: An augmented reality approach for computer supported collaborative work. *Virtual Reality*, 3(1):25-36, 1998.

[7] A. Butz, T. Höllerer, S. Feiner, B. MacIntyre, and C. Beshers. Enveloping users and computers in a collaborative 3D augmented reality. In *Proc. IWAR '99 (Int. Workshop on Augmented Reality)*, pages 35-44, San Francisco, CA, October 20-21 1999.

[8] M. Carpendale, D. Cowperthwaite, and F. Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Comp. Graphics and Applics.*, 17(4), 1997.

[9] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for Point-Feature label placement. *ACM Transactions on Graphics*, 14(3):203-232, July 1995.

[10] Y. Chrysanthou, and M. Slater. Computing dynamic changes to BSP trees, *Computer Graphics Forum (Proc. Eurographics '92)*, 11(3), September 1992, 321–332.

[11] E. S. Cohen, E. T. Smith, and L. A. Iverson. Constraint-based tiled windows. *IEEE Computer Graphics and Applications*, 6(5):35-45, May 1986.

[12] M. Deering and H. Sowizral. *Java3D Specification, Version 1.0*. Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA, Aug. 1997.

[13] K. Fairchild, S. Poltrock, and G. Furnas. SemNet: Three-dimensional graphic representations of large knowledge bases. In R. Guindon, editor, *Cognitive Science and its Applications for Human Computer Interaction*, pages 201-233. Lawrence Erlbaum, Hillsdale, NJ, 1988.

[14] S. Feiner, B. MacIntyre, M. Haupt, and E. Solomon. Windows on the world: 2D windows for 3D augmented reality. In *Proc. UIST '93 (ACM Symp. on User Interface Software and Technology)*, pages 145-155, Atlanta, GA, November 3-5 1993.

[15] S. Feiner, B. MacIntyre, and D. Seligmann. Knowledge-based augmented reality. *Communications of the ACM*, 36(7):52-62, July 1993.

[16] S. Feiner and D. Seligmann. Cutaways and ghosting: Satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer*, 8(5-6):292-302, June 1992.

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

[18] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Proc. SIGGRAPH '80*, pages 124-133, Seattle, WA, July 14-18, 1980.

[19] L. He, M. Cohen, and D. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and direction. In *Proc. SIGGRAPH '96*, pages 217-224, New Orleans, LA, August 4-9 1996.

[20] G. Hesina, D. Schmalstieg, A. Fuhrmann, and W. Purgathofer. Distributed open inventor: A practical approach to distributed 3D graphics. In M. Slater, editor, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-99)*, pages 74-81, N.Y., Dec. 20-22 2000. ACM Press.

[21] S. A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5-17, 1982.

[22] E. Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128-144, 1975.

[23] S. Julier, M. Lanzagorta, Y. Baillot, L. Rosenblum, S. Feiner, T. Höllerer, and S. Sestito. Information filtering for mobile augmented reality. In *Proc. ISAR '00 (Int. Symposium on Augmented Reality)*, pages 3-11, Munich, Germany, October 5-6 2000.

[24] T. Kamada and S. Kawai. An enhanced treatment of hidden lines. *ACM Transactions on Graphics*, 6(4):308-323, Oct. 1987.

[25] B. MacIntyre and S. Feiner. A distributed 3D graphics library. In *Computer Graphics (Proc. ACM SIGGRAPH '98)*, Annual Conference Series, pages 361-370, Orlando, FL, July 19-24 1998.

[26] M. Mine, F. Brooks, Jr., and C. Sequin. Moving objects in space: Exploiting proprioception in virtual-environment interaction. In *Proc. ACM SIGGRAPH '97*, pages 19-26, Los Angeles, CA, August 3-8 1997.

[27] T. Mori, K. Koiso, and K. Tanaka. Spatial data presentation by LOD control based on distance, orientation and differentiation. In *Proc. UM3 '99 (Int. Workshop on Urban 3D/Multimedia Mapping)*, pages 49-56, Tokyo, Japan, 1999.

[28] C. B. Phillips, N. I. Badler, and J. Granieri. Automatic viewing control for 3D direct manipulation. In M. Levoy and E. E. Catmull, editors, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 71-74, Cambridge, MA, Mar.-Apr. 1992. ACM Press.

[29] G. Robertson and S. Card. Fix and float: Object movement by egocentric navigation. In *Proc. UIST '97 (ACM Symp. on User Interface Software and Technology)*, pages 149-150, Banff, Alberta, October 14-17 1997.

[30] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[31] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. Picard, and A. Pentland. Augmented reality through wearable computing. *Presence*, 6(4):386-398, August 1997.

[32] R. Stoakley, M. Conway, and R. Pausch. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of Human Factors in Computing Systems (CHI '95)*, pages 265-272, May 7-11 1995.

[33] I. Sutherland. A head-mounted three dimensional display. In *Proc. FJCC 1968*, pages 757-764, Washington, DC, 1968. Thompson Books.

[34] Z. Szalavari, D. Schmalstieg, A. Fuhrmann, and M. Gervautz. Studierstube: An environment for collaboration in augmented reality. *Virtual Reality*, 3(1):37-48, 1998.

[35] W. Teitelman. A tour through CEDAR. *IEEE Software*, 1(2):44-73, April 1984.

[36] W. Thibault and B. Naylor. Set operations on polyhedra using binary space partitioning trees. C*omputer Graphics, 21(4), July 1987 (Proc. SIGGRAPH '87), pages 153–162.*

[37] J. van Roessel. An algorithm for locating candidate labeling boxes within a polygon. *The American Cartographer*, 16(3):201-209, 1989.

[38] S. You, U. Neumann, and R. Azuma. Hybrid inertial and vision tracking for augmented reality registration. In *Proc. IEEE Virtual Reality '99*, pages 260-267, Houston, TX, March 13-17 1999.