Proceedings of IEEE/ACM Int'l Symp. on Mixed and Augmented Reality (ISMAR 2003) (Tokyo, 7-10 Oct. 2003), pp. 66-75.

Evaluating Label Placement for Augmented Reality View Management

Ronald Azuma, Chris Furmanski
HRL Laboratories, LLC
{azuma, chris}@HRL.com

Abstract

View management, a relatively new area of research in Augmented Reality (AR) applications, is about the spatial layout of 2D virtual annotations in the view plane. This paper represents the first study in an actual AR application of a specific view management task: evaluating the placement of 2D virtual labels that identify information about real counterparts. Here, we objectively evaluated four different placement algorithms, including a novel algorithm for placement based on identifying existing clusters. The evaluation included both a statistical analysis of traditional metrics (e.g. counting overlaps) and an empirical user study guided by principles from human cognition. The numerical analysis of the three real-time algorithms revealed that our new cluster-based method recorded the best average placement accuracy while requiring only relatively moderate computation time. Measures of objective readability from the user study demonstrated that in practice, human subjects were able to read labels fastest with the algorithms that most quickly prevented overlap, even if placement wasn't ideal.

1. Motivation

One of the primary goals of Augmented Reality (AR) is to associate virtual information with their counterparts in the real world. While the virtual information is usually rendered in 3D, in some applications the virtual information may be confined to the 2D view plane. For example, virtual annotations rendered in a head-mounted display might be 2D labels identifying the names of nearby buildings. The decision of where and how to draw the annotations, to determine the spatial layout in the view plane, is the problem of *view management* [5]. This is a relatively new area of research in AR and Mixed Reality applications.

This paper focuses on a specific problem in view management: the placement and evaluation of 2D labels that are associated with 3D counterparts. A basic problem with drawing 2D labels is that the labels may obscure important background objects or may overlap, making them impossible to read. Figure 1 shows an example from our test application, with the labels arranged in randomly-chosen positions. This is a poor configuration because many labels overlap and are unreadable. Figure 2 shows a better labeling for exactly the same situation, where the labels were automatically repositioned by a cluster-based method described in Section 5.

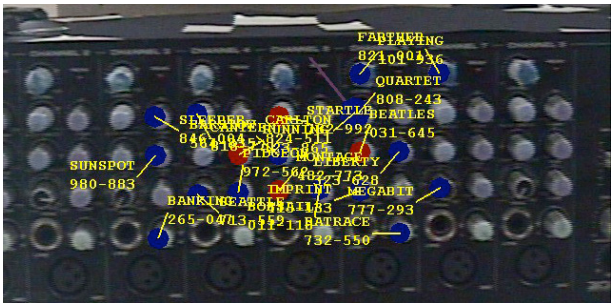

Figure 1: Initial (randomly chosen) label positions

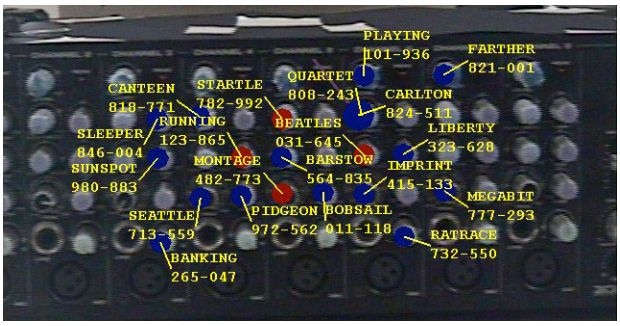

Figure 2: Repositioned labels through cluster-based method (red dial pattern #2)

Label placement for AR applications is not trivial. Even for a static image, the general label placement problem is NP-hard [7]. The number of possible label positions grows exponentially with the number of items to be labeled. For example, if a label can occupy one of 36 positions around an object, then for 20 objects there are 36^{20} possible label positions (over $1x10^{31}$). At interactive rates, only a few possible combinations can be considered. Furthermore, cognitive and perceptual issues regarding label placement for AR applications are not well understood. For example, if the label positions are not sufficiently temporally cohesive from frame to frame, the user might become distracted or impaired in the ability to read the labels [11]. Such factors require further study so that design guidelines can be generated for effective and practical label placement algorithms for interactive AR applications.

2. Previous Work and Contribution

Traditionally, label placement research has examined the problem of placing a large number of labels upon a static map, where computation time is not limited. The map-labeling bibliography [19] on the Internet has over 250 references; we will only discuss representative approaches and the most relevant examples. Several comprehensive surveys of the label placement problem exist [7] [8]. Map-labeling algorithms either perform exhaustive search or only search on a local basis [7]. Exhaustive search strategies, even with truncation heuristics, are too slow for interactive graphic applications except for very small examples. The exponential growth of the search space makes such an approach impractical. Instead, all interactive methods drastically limit the search space through various heuristics. Such approaches are often termed "greedy."

One common greedy approach is to consider each object in some order (e.g., based on priority) and set or modify the position of that object's label to the best location locally available [5] [23]. Another strategy is to model forces that "repel" adjacent labels from each other [12]. This strategy has been used in graph visualization, pushing the graph's nodes away from each other to maximize legibility. Many other heuristics exist. Greedy approaches are vulnerable to being trapped inside local minima. In contrast, simulated annealing [17] is a stochastic gradient descent algorithm that sometimes allows changes to the label configuration that make overlaps worse, rather than better. The likelihood of accepting an increase in overlaps decreases with time, according to an annealing schedule. The output of simulated annealing approaches the optimal solution as the computation time approaches infinity.

In the closest previous work to this paper, Bell et al. develop a general view management system for augmented and virtual reality applications that includes heuristics for improving temporal continuity of the labels from frame to frame [5]. This paper differs by focusing specifically on evaluating label placement, rather than more general view management issues. For example, the algorithms in this paper assume all labels *must* be drawn on the screen, even if overlaps result, so we can evaluate the algorithms consistently. In contrast, more general view management strategies would allow some labels to be removed through prioritization, aggregation, change in size, or transparency approaches. This paper directly and objectively compares several label placement strategies, including a version of the algorithm used in Bell et al.

The contribution of this paper consists of two main parts: 1) It offers the first evaluation of different label placement algorithms in an AR situation, including cognitive and perceptual issues and a user study. 2) It describes a new cluster-based label placement method and evaluates it against the other methods.

First, we evaluated four labeling algorithms using measurements of dynamic placement characteristics. Traditionally, label placement has been evaluated on static images with static criteria (ensuring visibility, unambiguously associating a label with only one object, aesthetic appeal, etc.) [26]. Evaluation in actual AR applications should also include dynamic characteristics such as label and scene motion. Section 3 describes how such dynamic characteristics can affect the human perceptual system. Sections 4 and 5 compare four label placement algorithms on motion data captured from an actual AR system, for an application specifically designed to test labeling (Section 6). Our evaluation consists both of collecting objective numerical measurements from the captured motion data (Section 7) and a user study directly comparing user performance in reading labels placed by the algorithms (Section 8). We also have video demonstrating the different methods operating in the AR test application.

The second contribution is a new cluster-based labeling algorithm. It was originally designed for an air traffic control (ATC) visualization application, and the output of an earlier version was shown in two previous papers [1][3], but those papers did not describe the algorithms and details of this method, as we do here in Section 5. The basic approach is to identify clusters: groups of labels and objects that overlap via a transitive relationship. These clusters represent the current problem areas for labeling. Then for each cluster, new sets of label positions are stochastically generated, where each new set changes the position of every label in the cluster simultaneously. This simultaneous repositioning enables this method to escape many local minima, which greedy methods tend to get stuck inside. It also considers the temporal aspects of the problem by avoiding moving labels unless required, maximizing the continuity of labels across iterations.

The concept of using clusters in a label placement strategy is not entirely new, although we have not seen this approach done before in interactive computer graphic applications. Non-interactive VLSI layout algorithms have broken up the task into user-defined clusters [1][18], but that is different from the automatic detection of clusters in a graphics application. The label detection papers on static maps that identify clusters use a *conflict graph* approach (e.g., [16]) that identifies clusters based upon where labels *could* overlap (if those positions were chosen), while the method described here computes clusters based upon labels and objects that *actually* overlap. A conflict graph approach performs a more thorough exploration of the search space and should provide more accurate solutions but requires more computation (e.g. one example required about one second of computation time for 30 nodes and 2-4 seconds for 60 nodes, on a SPARC 5 [16]) so they may be less suitable for interactive graphic applications. They also do not directly address temporal continuity issues.

3. Evaluation issues

Effective label placement algorithms must negotiate factors such as label visibility, position, and priority [5]; too many, unclear, or badly positioned labels may actually negate any benefits of augmenting the display because the viewer may become distracted or overloaded by the display. Human performance and AR/VR view management may be improved by experiments and *a priori* guidelines based on cognitive psychology [10].

The dynamic nature of AR drastically differentiates augmented video displays from other types of information labeling, such as cartography (i.e., static map-making). The need for real-time output constrains the design and implementation of AR systems. The movement of background and rendered images in AR displays are likely major contributors to the effectiveness of display design because of specific cognitive/perceptual limitations. So even though AR displays are intended to augment human abilities, human-centered approaches focusing on the cognitive affects of AR displays have received little attention. This section identifies what we believe to be the most important characteristics to measure and study, based upon existing knowledge from perception and cognitive science.

Dynamic AR displays involve several related aspects of human cognition including attention, eye movements, and working memory systems. More specifically, we believe the primary *a priori* cognitive issues unique to AR labeling readability, outside of static type-related characteristics such as size, contrast, and font, are the perceived motion and relative label distance, both of which may cause perceptual delays because of inefficient and inadvertent eye and attentional shifts.

Perceived motion: Attention is a core process of human cognition that actively selects and processes information from the environment or other internal mental processes [6]. Abrupt appearance and/or motion of objects in the visual field will automatically draw a person's attention to the location of the motion [14]. Voluntary shifts of attention can be made independently of eye movements [22], but automatic shifts of attention typically involve corresponding eye movements that are also automatically drawn to moving stimuli. This is especially relevant because the human perceptual system is most efficient at acquiring and processing visual information while the eyes are fixated (i.e., not moving) [20].

Automatic eye and attention shifts can occur even while viewers attempt to perform other attention-requiring tasks. When intentionally incorporated into displays, automatic shifts can serve as a useful alert or an aid for detecting novelty by drawing the user's attention to specific portions (e.g., suddenly appearing, flashing, or moving elements) of the display. However, abrupt stimulus onsets or movements can also be distracting to people, adversely affecting human performance by unintentionally diverting the user's attention away from the primary task. Such disassociations have been found in computer-based software applications with animation and moving icons that either inform or distract the user, depending on how the movement is used [4]. Possible strategies to avoid inadvertent gaze capture include rearranging the position of all labels simultaneously (possibly by incorporating the labels on an invisible surface that moves as a solid structure relative to the viewer) or moving labels when viewers move their eyes, so they won't notice the change (a perceptual phenomenon referred to as change blindness [24]).

Relative label distance: The relative separation/distance of the label from its corresponding object in the visual scene can also contribute to the overall latency of label reading. Human eye movements that follow lines between a rendered/virtual label and the corresponding real-world occur at a fairly constant rate, so scan time will linearly scale with distance (the eye movements called smooth pursuit movements that occur at a rate of roughly 50-70 degrees of visual angle per second [9]). Thus, the farther the labels are away from their corresponding object, the longer it will take the viewer to read the label.

Thus, relevant costs of optimizing label position include label motion, label-object separation distance, and readability (size, font, overlap, crowding/density). Because motion has such a strong effect on the human visual system, we hypothesized that reducing unneeded motion of AR tags was probably the most relevant factor in improving/reducing readability. Therefore, we decided to explicitly measure label motion (along with traditional measurements of overlaps and computation time).

Figure 3: Examples of individual label placements

A label placement is defined by an angle and radius away from the parent object. The text is drawn to the right or left of the endpoint of the label line, depending on which side the label is on (Figure 3). This improves readability by avoiding drawing the text over the label's own line. For this paper, we focus specifically on motion due to changing the angle, keeping the label radius constant (at 30 pixels).

4. Label Placement Algorithms

A previous survey of label placement on static maps [7] measured eight algorithms and concluded that three were most preferable, in order of increased computation time and solution quality: greedy depth first placement, discrete gradient descent, and simulated annealing. We adapted these three algorithms for AR label placement, along with our novel cluster-based method (Section 5).

We also implemented a force-based method, similar to those used in graph visualization, where nodes "repel" each other through distance-based forces. However, this

did not work well in preventing overlaps, and it often led to distracting oscillation problems where sets of adjacent labels would repeatedly push and be repelled from each other, like colliding bumper cars. Therefore this method is not included in this paper.

All methods were controlled and evaluated by one cost function that measures and penalizes for undesirable static label placements. This function attempts to penalize characteristics that make it difficult to read the labels or to easily associate an object with its label. It encodes an *a priori* guess of the factors that measure the "goodness" of a single label placement. Specifically, the cost function checks for labels covering each other, other objects, or other label lines, and for label lines crossing each other. An ideal cost is zero. Labels that cover active dial objects (the rendered blue or red circles) add 1 to the cost. A label that covers another label is penalized 10. A label that covers a label line carries a cost of 2, and a label line intersecting another label line adds 1 to the cost. The total cost sums the costs for all labels.

These tests are performed through straightforward geometric operations. For example, a label is seven characters wide by two characters tall, covering a box 70 by 30 pixels. A label covers another object (label, dial, line) if that any part of that object exists inside that box.

A label is considered to be in *overlap* with another label only if it covers another label. Covering another dial object or label line is not counted as an official overlap. This definition is important both for our measurements and the operation of some of the algorithms. Covering a dial object or label line or having lines intersecting could also be considered overlaps, but at the density used in the test application, the greedy, gradient descent and clustering algorithms all performed poorly with such a definition because it reduces the number of potential label locations that are free of overlaps. Therefore, the overlap definition is set this way to focus attention on the most basic problem: ensuring labels do not cover each other.

The initial default position of all labels (shown in Figure 1) was chosen randomly once, then used to initialize all the algorithms.

There is a tradeoff between how often label placements are computed and how smoothly labels transition between two settings. Computation costs aside, a new label placement could be computed for each frame. However, tests with pilot stimuli showed that if a label angle changes significantly, this "jump" in location caused delays and errors as users have to search for the new label position. Instead, we performed label placement at 2 Hz. Since the rendering occured at 20 Hz, the intermediate frames are used to linearly interpolate the label from the initial position to the new position. The rotation direction is picked to minimize the angle traversed. This allows the user to follow the label from the old to the new position; however the labels may not be at appropriate locations during the intermediate or final frame (since the underlying scene can change as the user changes the viewpoint). In our test application, the viewpoint motion was relatively slow and coherent, so the 2 Hz update rate was an acceptable compromise.

We now briefly describe the greedy, gradient descent and simulated annealing algorithms. The new cluster-based method is explained in the next section.

Greedy algorithm: This is a depth first search with no backtracking that attempts to perform the minimum amount of work possible. At startup, the initial angles are randomly shuffled. The 20 dials are placed one by one in a priority order, and this order is randomly shuffled before each placement. For each dial, we check if its label is in overlap. If not, the label remains at its original angle. If it overlaps, then we search the label angles from 0 to 350 degrees in steps of 10 degrees and accept the first one that has no overlaps. If no such solution exists, the label remains at its original angle.

Gradient descent: Our version is similar to the gradient descent approach in Bell et al. [5]. At startup, the initial angles are randomly shuffled. The 20 dials are placed one by one in a priority order, and this order is randomly shuffled before each placement. For each dial, we examine 36 angles (0-350 degrees in steps of 10) and find the one that is furthest from all other labels and active dials while also not being in overlap. We perform this test since we did not implement the space management code in Bell et al. We implemented the hysteresis used in Bell et al. to reduce label motion. We compute both the ideal location and the closest match to the previous location. If those do not match, we use the closest and eventually switch to the ideal after a counter exceeds a time limit (two seconds). A true gradient descent repeats the examination of all dials until no more improvement is possible, but to keep this at interactive rates we only examine each dial once per placement.

Simulated annealing: We used a fast implementation called Adaptive Simulated Annealing (ASA) [13] that is publicly available at www.ingber.com. Since the annealing algorithm and cost metric are different than the one in Christensen et al. [7], we could not use the specified annealing schedule described there. Instead, we experimented with the settings to find one that tended to find a zero cost arrangement for every placement while completing within a few seconds. The initial positions provided to the algorithm are the ones from the previous placement; if that position already has zero cost then ASA will not move the labels. Otherwise, there is nothing particular in ASA that minimizes label motion. Simulated annealing generally requires more time to complete than can be supported at interactive rates; it is included here to provide a comparison against a method that produces nearly ideal placements at all times (but at the cost of significant label motion and CPU time).

5. Cluster-Based Method

The general strategy of our new cluster-based method is to identify clusters of objects that mutually overlap, thus

allowing the algorithm to search for solutions by simultaneously repositioning all the labels in a cluster. This avoids being trapped in some local minima that many other methods get stuck in because they only move one label at a time. In many practical applications, such as the ATC visualization example, overlap problems are naturally grouped into such clusters. This provides a strategy for productively constraining the search space to attack the areas that cause the most problems.

Picking the new label locations proceeds in three steps: computing the label overlaps (i.e., finding where the problems are), identifying the clusters, and exploring the search space on a cluster by cluster basis.

Step 1 (Computing overlaps): The algorithm computes the cost of the current label configuration and identifies which labels overlap others, as defined in Section 4.

Step 2 (Identifying clusters): A cluster is defined as a group of objects that transitively overlap each other. For example, say that object *A* overlaps object *B*, and object *B* overlaps object *C*, but object *A* does not overlap object *C*. Then *A*, *B*, and *C* all fall into one cluster. The algorithm computes clusters by clearing an initial list of clusters and then iterating through each object. If the object has overlaps (determined by the overlap list from Step 1), then it determines if this object already exists in a cluster or not. If it does, then the list of objects that overlap this object is added to the cluster list. Otherwise, a new cluster is created and those overlapping objects are put into the new cluster list, along with this object. After this insertion is done, objects may exist in multiple clusters. Therefore, the algorithm checks the cluster it just inserted into (or created) against all other clusters. If there are any duplications of object ID's, then those clusters are merged together. After all the iterations are done, the list of clusters is sorted by the number of objects in each cluster, so that the largest clusters are at the start and the smallest at the end. This makes the algorithm attack the largest clusters first. Note that the cluster lists may not include all objects. If an object does not overlap another object and it itself is not overlapped by another object, then it will not exist in a cluster and its label position will not be moved. This is important because this avoids moving labels that are not currently causing any problems.

Step 3 (Exploring the search space): This step initializes the new position of each label to be the position from the previous iteration. The algorithm examines each cluster, in order from largest to smallest. For each cluster, several new sets of label positions are chosen and evaluated for all the objects in that cluster. These parameters are adjustable, but in the current implementation, the number of sets is 40 if there are 2 or 3 objects, and 75 for four or more objects. One of the sets has all the labels in a default position (upper right corner), and another set has all the labels in their current position. This ensures that the algorithm will not choose a new solution that has a greater cost than the current solution. The remaining sets are chosen stochastically. For each object, the proposed new angle is chosen randomly. Therefore, the algorithm escapes some local minima by moving all the labels in a cluster simultaneously, rather than just one label at a time. Each set of label positions is then evaluated for a total penalty cost. This cost is the sum of all penalty costs computed for each object in the cluster, as computed by the routine described in Step 1. The algorithm selects the set with the lowest overall cost and that defines the new positions of the labels for all objects in that cluster.

The limited number of sets that are searched guarantees that this algorithm completes in a reasonable amount of time. But it also means that it may not find a good solution in this iteration. However, if the underlying objects stay in the same position, as time progresses this algorithm has a greater likelihood of finding a good solution. The algorithm spreads out the computation load over time, preserving real time interaction while progressively refining the solution.

Because the cluster method simultaneously repositions multiple labels that have been identified as being linked together, it is able to escape some local minima that other greedy methods cannot improve. However, this method does not avoid all local minima. For example, if a cluster of problem labels is surrounded by a ring of labels that are not in overlap, then there is no room to move the problem labels, even if the situation is globally solvable by moving some labels in the non-overlapping ring.

6. Testbed

The label placement algorithms were tested and demonstrated on an AR system, which consisted of a PC with Windows 2000, two Xeon 1.7 GHz CPU's, an NVIDIA Quadro4 900 XGL graphics board, a Matrox Meteor II frame grabber, and a National Instruments PCI-MIO-16XE-50 A/D converter. It was a video see-through system, with video provided by a Toshiba IK-SM43H pen camera with an 8mm lens. We used the 3rdTech Hiball-3000 optical tracker. The real object to be labeled was an audio mixer with an array of knobs and dials. The background behind the mixer was colored black to ensure the labels were easily readable. Figure 4 shows a view of the equipment.

Figure 4: Audio box viewed by HiBall-tracked sensor box

We chose a set of 20 randomly picked dials to label, which stays the same across all motion sequences. Twenty labels was an appropriate density, dense enough to be difficult for the placement algorithms and human viewers but sparse enough to allow readable solutions. If the density was low, then every placement algorithm does well. If the density was too high, then computation time becomes intolerably large, so it may be more effective to switch to different view management techniques such as transparency, filtering and aggregation [15]. One of the virtual dials was placed 0.2 meters in front of the plane of the other dials (Figure 5). This was done so that motion parallax effects would cause that dial to move around in front of the other dials, forcing labels to change positions.

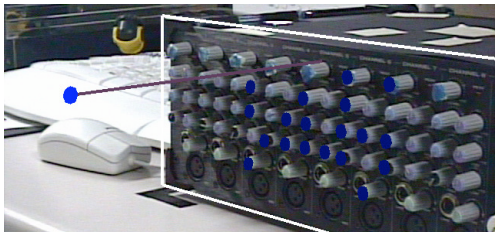

Figure 5: Oblique view of the 20 virtual dials (in blue)

The onboard PC clock had only a 10 ms resolution, so for accurate timing we used the A/D board's clock, which had a 10 microsecond resolution.

The label placement algorithms (other than ASA) could run interactively, but to perform a controlled experiment the algorithms had to operate on exactly the same motion sequence. Since users could not exactly replicate real motion across several different runs, we added the ability to record both tracking data and background images to RAM, which at the end of the run are saved to disk. This allowed each algorithm to be run offline on the same prerecorded motion. The initial label starting positions were the same for all trials and algorithms.

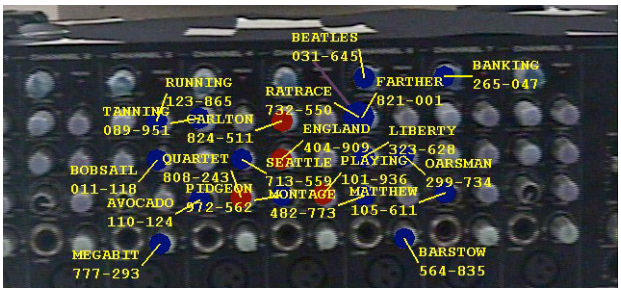

Figure 6: Greedy method (red dial pattern #0)

We collected two motion sequences. In the first, the viewing camera panned horizontally, rotating to keep the center in view. In the second, the camera moved vertically. Each sequence lasted 20 seconds. Examples of placements generated by the four algorithms for the horizontal motion sequence are in Figure 2 and Figure 6 through Figure 8. Those four images also show the four different patterns of red dials that were the focus of the user study. The algorithms did not attach any priority to the red dials or treat them differently than the blue dials.

Figure 7: Gradient descent (red dial pattern #1)

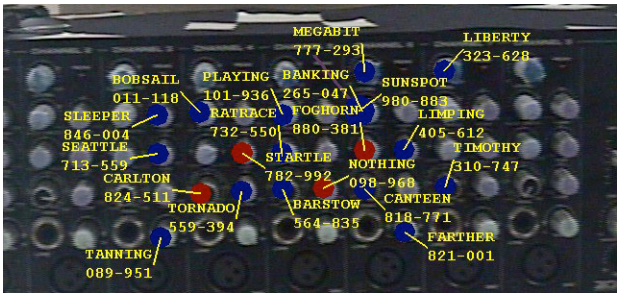

Figure 8: Adaptive simulated annealing (red dial pattern #3)

7. Numerical Evaluation

We ran each algorithm eight times, four on the horizontal motion sequence and four on the vertical motion sequence (one for each red button pattern). The playback program computed several performance metrics on these eight runs, summarized in the following tables. Each value is given for each label placement operation.

Algorithm	Min	Max	Median	Mean
Greedy	1.1	10.3	3.7	3.6
Clustering	2.1	48.4	7.2	8.4
Gradient descent	37.3	43.9	37.9	38.0
ASA	762	5561	3663	3515

Table 1: Computation time in milliseconds

Algorithm	Min	Max	Median	Mean
None	366	558	490	482.5
Greedy	15	115	53	51.7
Clustering	0	267	6	13.4
Gradient descent	7	140	30	36.3
ASA	0	24	0	2.1

Table 2: Cost

Algorithm	Min	Max	Median	Mean
None	16	17	17	16.8
Greedy	0	4	0	0.97
Clustering	0	10	0	0.55
Gradient descent	0	8	0	0.95
ASA	0	2	0	0.03

Table 3: Number of overlaps

Algorithm	Min	Max	Median	Mean
Greedy	0	6	0	0.54
Clustering	0	17	2	2.2
Gradient descent	0	14	1	1.8
ASA	0	20	19	12.0

Table 4: Number of labels moved

Algorithm	Min	Max	Median	Mean
Greedy	0	1	0	0.02
Clustering	0	5	1	0.82
Gradient descent	0	11	1	1.08
ASA	0	20	15	10.4

Table 5: Number of labels moved that were not originally in overlap

Computation time analysis: Table 1 summarizes the computation time measurements. Greedy was faster than clustering, which was faster than gradient descent. Greedy was typically much faster with less dense situations (well under one ms); it slowed down significantly given the density in our test application. Note that our implementation for computing overlaps was brute force, involving $O(N^2)$ comparisons. More intelligent approaches that partition the view space [5] could speed up all the algorithms significantly. However, since we shared the same cost/overlap computation subroutine amongst all the label placement algorithms, this provided a consistent basis for comparison.

Placement accuracy: All the tested algorithms significantly improved the situation from the initial condition. Whether measured by the *a priori* cost function (Table 2) or by simply counting the number of overlaps (Table 3), clustering had better average placement scores than greedy or gradient descent. The median number of overlaps was zero for all algorithms but clustering's mean was about half of greedy or gradient descent. ASA was almost always perfect.

Motion: Table 4 and Table 5 present the motion statistics. Greedy, clustering and gradient descent all attempted to minimize label motion. Gradient descent typically pushed the labels on the edge outward, which was beneficial to placement as it left more room for the other labels in the middle. Clustering and gradient descent moved more labels than greedy. ASA moved almost all the labels most of the time, even though nearly all were not in an overlap.

The algorithms have different "settling" characteristics. Placement can be analyzed in two phases: startup (when the placement algorithms work to find a good placement from the poor initial position) and maintenance (after a good placement has been achieved and changes result mostly from motion parallax causing the dial in front to move around). Both greedy and ASA have a short startup phase, while clustering and gradient descent have long startup phases. Greedy typically reaches its best placement almost immediately and only changes it minimally thereafter; clustering and gradient descent work more slowly but ultimately achieve better placements, as measured by the cost function and counting overlaps. This is discussed in more depth in Section 9.

Summary: Of the three algorithms that can run in real time (greedy, clustering and gradient descent), clustering recorded the best average placement accuracy while falling in the middle in required computation time. However, clustering tended to move more labels. ASA resulted in almost perfect placements, but it was too slow for real-time operation and caused much unneeded motion. Greedy and ASA had the shortest "startup" times.

It should be noted that all these algorithms are stochastic, so even with exactly the same motion sequence and set of initial conditions, each execution of an algorithm will generate a different output. To estimate how much the output can vary, we used the horizontal motion sequence and ran that twelve times each on the greedy, clustering and gradient descent methods. Table 6 lists the mean and the standard deviation of the median values for computation time and cost in those twelve executions, where standard deviation is in parentheses after the mean.

Algorithm	Computation Time, in ms	Cost
Greedy	3.0 (2.3)	49.7 (17.0)
Clustering	6.9 (1.5)	7.2 (4.5)
Gradient descent	38.0 (0.07)	31.9 (8.4)

Table 6: Mean and (std. dev.) of the medians

Except for gradient descent's computation time, there was significant variation between executions. However, this variation was not large enough to remove the differences between the mean results.

8. Empirical Evaluation (User Study)

The goal of this experiment was to find which of the algorithms allowed human users to best (most quickly) read AR data tags. By measuring the reaction times of people reading AR labels, we hoped to experimentally validate that user performance was well correlated with certain algorithms. We expected that the most successful algorithms (those that presented the most readable text in the most expeditious manner) would produce the shortest reaction times measured in this empirical user study. We hypothesized that algorithms with excessive movement (e.g., ASA) might be the most distracting, and hence, produce the slowest reaction times.

Stimuli: Stimuli were digitized movies made from captured video frames recorded in the AR system described in Section 6. Video images were of a multi-dial piece of audio equipment, shown in Figure 1, that was augmented with computer rendered tags and dials. Each movie segment was either a vertical or horizontal back-and-forth pan lasting 20 seconds. Movies were presented in a window sized at 640x480 pixels, subtending approximately 24.2 x 19.9 degrees of visual angle. Movies were captured and played at 20 frames per second.

Virtual objects that augmented the display consisted of 19 dial faces that appeared in the same depth plane as the actual dial face, and one "occluding" dial was rendered

20cm closer to the camera/viewer (see Figure 5). Because this "occluding" dial was rendered in a depth plane closer to the viewer, the panning movement caused motion parallax that caused ongoing label replacement.

Design: This experiment tested the effectiveness of four types of algorithms in a classic within-subject design (each subject experienced all conditions). Each trial in the experiment was defined by the algorithm type [Adaptive Simulated Annealing (ASA), Greedy (G), Gradient Descent (GD), Clustering (C), and None (N)], panning direction [Vertical (V), and Horizontal (H)], and four patterns of red target buttons (see Figure 2 and Figure 6 - Figure 8), for a total of 40 unique trial types.

Across all trials, the presentation order of each stimulus was counterbalanced using a modified Latin-Square (a method of balancing the distribution of trial types to guarantee each trial type occurred equally often following any other trial type).

The experiment consisted of 2 practice trials (used to familiarize the subjects with the task; these trials were not included in the analyses), followed by 50 experimental trials (all 40 of the unique trial types plus 10 trials that repeated the first 10 trials in order to complete the counterbalancing ordering). Since the key experimental factor was the algorithm type, varying the panning direction and button patterns simply added diversity in an attempt to prevent boredom in subjects. In the final analysis, we collapsed across button pattern and panning direction, so that we collected 10 repetitions of each algorithm condition (ASA, G, GD, C and N). While there was a difference in mean reaction time between the vertical and horizontal motion trials, this collapse did not affect the overall pattern of effects.

Procedure: Subjects were instructed to read aloud the top line of each data tag associated with a red target dial. There were four red dials in each stimulus. For each trial, the data tags were selected randomly without replacement from a pool of thirty 7-letter words. Subjects were instructed to perform the task as quickly as possible, while maintaining the highest level of accuracy, thus focusing on accuracy instead of speed.

Each trial was initiated with the subject's key press and terminated by another key press, when they had finished reading the last label. The PC recorded the reaction time. Experimenters scored the correctness of each trial by hand. Feedback was not provided.

Participants were 6 members of the research staff with little or no exposure to stimuli nor extensive interaction with AR displays.

Analysis: Reaction times (RT) were collected for each subject across the 50 experimental trials. RT were averaged together for each subject, for each condition for only those trials that were answered correctly (by correctly reading all four labels). Also, the first 10 trials (2 trials for each condition) were excluded to reduce the variance cased by practice (the first two trials were on average 25% slower than the following 8 trials). Mean RT for subjects, averaged across all correct trials and all conditions, ranged from 5.95s to 8.17s (mean 7.15s, SD=0.78s). In order to increase the statistical power, between subject variability was reduced by normalizing (dividing) all of the RT for each subject by the mean RT of the fastest condition. In all cases, ASA was the fastest condition (mean ASA RT were 5.32s, SD=1.03s).

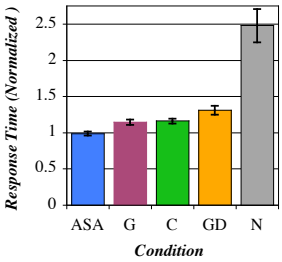

Figure 9: Normalized response times (seconds) for 6 subjects for 5 algorithm patterns

Results: The carefully-designed within-subjects design, numerous trials, and robust differences in reaction times led to reliable results despite the small sample size (6 subjects). Figure 9 displays the reaction times for the five different algorithm conditions for all subjects. Bars are the mean RT, averaged across 6 subjects after normalizing each subject's RT by his or her own mean ASA RT (see previous paragraph titled *Analysis* for rationale). Error bars for all plots are +/- one standard error (STD/ n). Planned pairwise T-tests were performed for the statistical analyses presented below.

These data indicate that subjects were reliably fastest for reading labels with the ASA algorithm ($p<0.0005$, $t(46)=3.83$), and reliably slowest for no (N) placement algorithm ($p<<0.0001$, $t(46)>>100$). GD was reliably slower than ASA, G, and C ($p<0.05$, $t(46)=2.19$). There was a trend for G to be faster than C, but the difference was not reliable ($p>0.05$, $t(46)=0.32$).

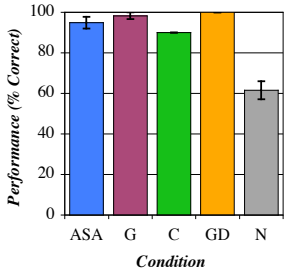

Figure 10: Task performance for 6 subjects for 5 algorithm patterns

Figure 10 shows the performance (%correct) for the different conditions across all subjects.

9. General Discussion

At first glance the patterns of RT in Figure 9 were surprising, for several reasons. First, we expected a parallel between the behavioral measures of RT plotted in

Figure 9, and some of the metrics outlined in Table 1 through Table 6. There was no clear relationship between our calculated cost or label-movement metrics. Second, because we thought motion would be a major factor, we expected algorithms that contained the most movement, such as ASA, to yield the slowest RT. As it turned out, ASA, the algorithm that had the most total motion, resulted in the fastest RT.

Several alternative hypotheses are feasible. For example, our RT measure might not have been sensitive enough to appropriately capture motion's effect on reading speed.

It seemed conceivable that other factors besides motion may have played a bigger role in altering subjects' behavioral performance. Furthermore, the numerical evaluation in Section 7 took all 20 dials/labels into account for the entire 20 seconds of recorded data, while the perceptual reading task only dealt with a subset of four dials and subjects typically completed the task within 7 seconds. Therefore, we reanalyzed the numerical analysis, focusing on the number of overlaps that occurred for just the four target labels across the first 10 seconds of each trial (the mean RT for all conditions was about 7.2 seconds).

Figure 11 shows the average number of label overlaps involving only the labels for the 4 red dials for each algorithm plotted as a function of time. On average, the fact that C and GD continued to have overlapped labels within the first 5 seconds of the trials suggests that overlapping labels may play the dominant role in how quickly subjects could read all the labels.

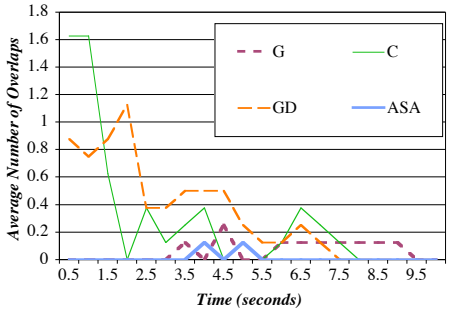

Figure 11: Number of label overlaps for only the target (red) dials plotted as a function of time for 4 algorithms

Figure 12 plots the average number of label overlaps for all labels (not just the ones for the 4 red dials), for the entire 20 seconds. For the majority of the sequence (after the 3 second mark), C usually had fewer average overlaps than G or GD. This explains why C had better scores in placement accuracy than G or GD as measured in Section 7. However, G and ASA had a fewer average number of overlaps during the first few seconds. GD and C have a much longer startup phase than G and ASA. During startup, both GD and C created more overlaps but eventually found a better overall placement (as scored by average number of overlaps or the cost function) than G.

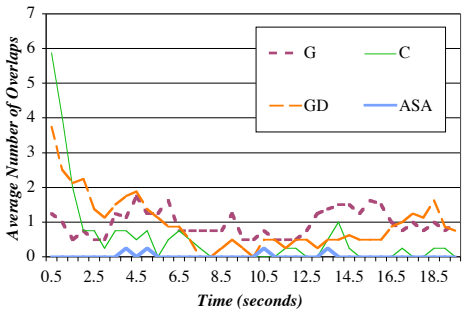

Figure 12: Number of label overlaps for all dials plotted as a function of time for 4 algorithms

If overlaps are the major determining factor in the readability of labels, then this suggests that the choice of label placement strategy may depend on what the AR application does. If an application generates situations similar to the "maintenance" phase (e.g. when the user studies an object, does not translate much, or the real objects are far away, as in an outdoor application), then perhaps clustering is a good approach. However, if the AR application spends most of its time in the "startup" phase (e.g., the user viewpoint or real world situation often changes dramatically), then perhaps the better strategy is embodied in the greedy method: reach a "best guess" quickly without looking for better overall solutions.

10. Future Work

Traditionally, label placement research focused on static mappings: placing labels once over a static background. Such research has been applied to labeling maps that are generated dynamically, but there may not be any similarities between the current map and the next, and generally a few seconds are available to label each map [21]. In contrast, interactive computer graphic applications are a new application area for label placement algorithms, where the 2-D screen positions of the objects to be labeled often exhibit strong temporal coherence from frame to frame. This coherence and interactive update rate suggest areas for future work.

First, a series of experiments could be run to more specifically examine and quantify the individual cognitive and perceptual issues associated with placing and moving labels. Topics include line crossing, label line length, motion-based distraction, and at what rate placements should be performed. Motion may affect other aspects of AR applications not measured by our user study. Eventually these could result in a new, validated cost metric that can guide the development of new placement algorithms for AR applications.

Second, there is an opportunity to develop new algorithms that take advantage of the temporal coherence inherent in interactive applications. For example, a label placement method might identify better solutions by

considering not just where the labels should go in the current frame but also where they should be in future frames. If the application can accurately predict the future positions of the objects to be labeled, it could optimize label placement across the entire motion sequence. One such strategy would be to identify an initial configuration that reduces the number of labels that have to be moved. However, this approach assumes accurate prediction of the future state (including head, hand and object locations) and requires much more computation. Such an approach would have to be tuned to a particular application to be effective. For example, predicting the future state of aircraft in an ATC application is different from predicting head and object motion in an AR application.

Third, a user can only read a few labels at any given instant. If the system knew exactly which ones those were, perhaps through eye tracking, a label placement algorithm could set priorities on certain labels to ensure those are easily readable, even at the cost of worse placements on the other labels.

11. Acknowledgments

Raytheon Company funded the development of the cluster-based approach. We thank David Bloomstran, Gene Opittek, Fred Messina, Thomas Briere, Ken Arkind and Ed Stevens of Raytheon Company for their support. Howard Neely, Ron Sarfaty and Luis Murdock helped with the video recording. Mike Daily provided guidance and support.

12. References

[1] Areibi, S., Thompson, M. and Vannelli, A. A Clustering Utility Based Approach for ASIC Design. *Proc. IEEE ASIC/SOC Conf.* (2001, Arlington, VA), pp. 248-252.

[2] Azuma, R., Neely III, H. Daily, M., and Correa, M.. Visualization of Conflicts and Resolutions in a "Free Flight" Scenario. *Proc. IEEE Visualization* (24-29 Oct. 1999, San Francisco), pp. 433-436.

[3] Azuma, R., Neely III, H., Daily, M., and Geiss, R.. Visualization Tools for Free Flight Air-Traffic Management. *IEEE Comp. Graph. & Apps 20*, 5 (Sept/Oct 2000), pp. 32-36.

[4] Bartram, L., Ware, C., and Calvert, T. Moving Icons, Detection and Distraction. *Proc. Interact 2001*, (9-13 July 2001, Tokyo).

[5] Bell, B., Feiner, S., and Höllerer, T. View Management for Virtual and Augmented Reality. *Proc. Symp. on User Interface Software and Technology* (11-14 Nov. 2001, Orlando, FL), pp. 101-110.

[6] Broadbent, D. E. (1958). Perception and Communication. London: Pergamon.

[7] Christensen, J., Marks, J. and Shieber, S. Labeling Point Features on Maps and Diagrams. Technical report TR-25-92, Center for research in computing technology, Harvard University, 1992. http://www.eecs.harvard.edu /~shieber/papers-date.html

[8] Christensen, J, Marks J., and Shieber, S. An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Trans. on Graphics 14*, 3 (July 1995), pp. 203-232.

[9] Eckmiller, R., and Bauswein, E.. Smooth pursuit eye movements. *Progress in Brain Research 64* (1986), pp. 313-323.

[10] Furmanski, C, Azuma, R, and Daily, M. Augmented-reality visualizations guided by cognition: Perceptual heuristics for combining visible and invisible information. *Proc. IEEE/ACM Int'l Symp. Mixed and Augmented Reality*. (30 Sept. – 1 Oct. 2002, Darmstadt, Germany), pp. 215-224.

[11] Granaas, M., McKay, T.D., Laham, R.D., Hurt, L.D., Juola, J.F. Reading Moving Text on a CRT Screen. *Human Factors 26*, 1 (1984), pp. 97-104.

[12] Hirsch, S.A. An Algorithm for Automatic Name Placement Around Point Data. *The American Cartographer 9*, 1 (1982), pp. 5-17.

[13] Ingber, L. Very Fast Simulated Re-annealing. *Mathematical Computer Modelling 12* (1989), pp. 967-973.

[14] Jonides, J., and Yantis, S. Uniqueness of abrupt visual onset in capturing attention. *Perception and Psychophysics 43* (1988) pp. 346-354.

[15] Julier, S., Lanzagorta, M., Baillot, Y., Rosenblum, L., Feiner, S., Höllerer, T., Sestito, S., Information Filtering for Mobile Augmented Reality. *Proc. IEEE/ACM Int'l Symp. Augmented Reality* (5-6 Oct. 2000, Munich), pp. 3-11.

[16] Kakoulis, K.G. and Tollis, I.G. A Unified Approach to Labeling Graphical Features. *Proc. ACM 14th Symp. Computational Geometry* (7-10 June 1998, Minneapolis, MN), pp. 347-356.

[17] Kirkpatrick, S., Gelatt Jr., C.D., and Vecchi, M.P. Optimization by Simulated Annealing. *Science 220* (May 1983), pp. 671-680.

[18] Mallela, S. and Grover, L.K. Clustering Based Simulated Annealing for Standard Cell Placement. *Proc. Design Automation Conference (DAC '88)* (1988, Los Alamitos, CA), pp. 312-317.

[19] The Map-Labeling Bibliography. http://www.math-inf.uni-greifswald.de/map-labeling/bibliography/

[20] Matin, E. Saccadic suppression: a review and an analysis. *Psychological Bulletin 81* (1974), pp. 889-917.

[21] Petzold, I., Plümer, L. and Heber, M. Label Placement for Dynamically Generated Screen Images. *Proc. 19th Int'l Cartographic Conference* (ICA '99) (1999, Ottawa, Canada), pp. 893-903.

[22] Posner, M.I. Orienting of attention. *Quarterly Journal of Experimental Psychology 32* (1980), pp. 3-25.

[23] Pritt, M. Method and apparatus for the placement of annotations on a display without overlap. US patent 5,689,717 (Nov. 18, 1997).

[24] Simons, D. J. Current Approaches to Change Blindness. *Visual Cognition 7* (2000), pp. 1-15.

[25] Strijk, T, and van Kreveld, M. Practical Extensions of Point Labeling in the Slider Model. *Proc. 7th Symp. Advances in Geographic Information Systems* (5-6 Nov. 1999, Kansas City, MO), pp. 47-52.

[26] van Dijk, S., van Kreveld, M., Strijk, T., and Wolff, A. Towards an Evaluation of Quality for Label Placement Methods. Technical Report UU-CS-2001-43. Dept. of Computer Science, Utrecht University, 2001.