Bubble Clusters: An Interface for Manipulating Spatial Aggregation of Graphical Objects

Nayuko Watanabe[†] *Motoi Washida*[‡] *Takeo Igarashi*[‡]

[†]Research & Development Center
Toshiba Corporation
nayuko.watanabe@toshiba.co.jp

[‡]Computer Science Department
The University of Tokyo, SORST JST
wm3@ui.is.s.u-tokyo.ac.jp, takeo@acm.org

ABSTRACT

Spatial layout is frequently used for managing loosely organized information, such as desktop icons and digital ink. To help users organize this type of information efficiently, we propose an interface for manipulating spatial aggregations of objects. The aggregated objects are automatically recognized as a group, and the group structure is visualized as a two-dimensional bubble surface that surrounds the objects. Users can drag, copy, or delete a group by operating on the bubble. Furthermore, to help pick out individual objects in a dense aggregation, the system spreads the objects to avoid overlapping when requested. This paper describes the design of this interface and its implementation. We tested our technique in icon grouping and ink relocation tasks and observed improvements in user performance.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces – Graphical user interfaces.

General terms: Design, Human Factors

Keywords: spatial layout, direct manipulation, clustering

INTRODUCTION

In current graphical user interfaces, spatial layouts are often used to manage temporary or transient information. Examples include desktop icons, web bookmarks [18], and spatial hypertexts [21]. In these systems, the spatial position of objects represents the semantic relationships among them. Objects placed close together are related and objects placed far away from each other are unrelated. Compared to more explicit grouping mechanisms, such as hierarchical folders, these spatial layouts enable more flexible and lightweight organization of information. They are particularly useful for managing miscellaneous personal information [2].

Spatial aggregation plays an important role in management of information using spatial layout. An aggregation represents a group structure and people can quickly create, merge, and split groups by moving objects on the screen. However, in most existing systems, a group represented by a spatial aggregation exists only in the user's mind, and the computers do not explicitly treat the group as a group. Therefore, the user needs to explicitly select the objects in a group using methods such as rubber banding or lasso selection before applying an operation (e.g. relocation or deletion) on the group. Another problem is that it is difficult to pick a hidden object from a dense spatial aggregation. To pick such an object using existing systems, the user needs to remove overlapping objects individually.

We propose an interface, called Bubble Clusters, for manipulating spatial aggregations of objects to address these problems. First, the system automatically recognizes aggregated objects as a group and visualizes the result as a bubble surrounding the objects (Figure 1). The bubble serves as a proxy for the group and the user can operate on the group by clicking on the bubble. Unlike standard folders, our system does not require an explicit grouping operation. The user simply creates, deletes, and moves objects, and the system automatically updates the group structure according to spatial relationships. Second, the system supports automatic spreading of aggregated objects to help the user to pick a hidden object. An aggregation expands temporarily at the user's request (by double-clicking on the bubble) and automatically returns to its original state once the user has picked a target object or cancelled the operation.

In addition to being visually pleasing, the bubble metaphor is particularly suitable for our purposes for the following reasons. First, the rotund shapes present themselves as transient structures, encouraging further experiments in organizing documents [15]. Second, we can leverage users' natural expectation of bubble behavior to help them to easily understand complicated behavior, such as the hysteresis effect that we describe later. Third, a bubble shape is spatially efficient in comparison to a box [15] or a convex hull [2] because it compactly encloses the target objects.

This paper describes the design of our interface and its implementation details. We refined the design through iterative design processes, and the resulting interface contains many subtle, yet highly effective details. These include a hysteresis effect in the merger and splitting of bubbles.

UIST'07, October 7-10, 2007, Newport, Rhode Island, USA.

This paper also introduces possible applications of the technique and reports the results of a user study where we compared Bubble Clusters with standard folders in an icon grouping task and with lasso selection in an ink relocation task. We found that Bubble Clusters can improve user performance by reducing the amount of user operations.

RELATED WORK

Spatial Layout and Aggregation

Spatial layout has long been used for the management of documents in computing systems. A spatial layout of icons was used in the early Macintosh and it is one of the fundamental features of current window systems such as Windows and Mac OS. Several extensions have been made to these simple 2D layouts, such as zoomable interfaces [3, 16] and a 3D arrangement [18].

Additional methods for controlling the spatial aggregation of icons have also been proposed. Mander et al. observed that people create piles of documents to organize information in the real world, and developed a gestural interface for interacting with piles [13]. Dynapad extends zoomable desktops and introduces open, self-adjusting clumps for managing groups of icons [2]. The BumpTop system uses physics simulation to add realism to the behavior of desktop icons [1]. This enables users to create various group structures, ranging from tidy piles to messy aggregations. Our system differs from these systems in that they require the user to explicitly select multiple objects using lasso selection or rubber banding to initiate a new clump or a pile. In contrast, a bubble cluster automatically emerges whenever objects are aggregated. We believe that these two approaches will complement each other. Bubbles are suitable for transient grouping while explicit clumps or piles are suitable for more permanent groups.

Object Manipulation and Group Manipulation

Object selection and manipulation are some of the most fundamental operations in graphical user interfaces and many techniques have been proposed to improve them. An area cursor [22] and bubble cursor [8] make the mouse cursor larger. This makes it easier to catch a small target in a relatively sparse layout. In contrast, Bubble Clusters is designed to make object selection in dense layouts easier. Saund et al. proposed overloaded mouse drag selection, which combines rubber banding and lasso selection [19]. The user can use rubber banding by drawing a diagonal stroke and seamlessly switch to lasso selection by drawing an enclosure. They also proposed a flat lattice grouping structure [20] to enable an object to belong to multiple groups. The user can cycle through the groups by successive clicks.

Some systems attempt to infer implicit grouping structures in object layouts. Shipman et al. analyzed typical spatial structures and developed algorithms to detect these common structures [21]. Igarashi et al. extended their work by focusing on less structured layouts [9]. Advanced grouping mechanisms for handwritten strokes have also been researched in the context of electronic whiteboard and drawing systems [11, 14, 15, 20]. Our method frees users from explicit grouping and makes implicit group structures visible by using a sophisticated visual representation to more actively support the spatial organization task.

INTERFACE DESIGN

This section describes the behavior of Bubble Clusters from the user's point of view. We first discuss the visual representation of bubble clusters and then describe their behavior.

Visual Representation

The base application that we use in this explanation is a simple object manipulation system. The user can freely move and arrange small objects on a 2D flat canvas using a standard direct manipulation (i.e. drag-and-drop) interface. This application is an abstraction of typical object manipulation tasks in current GUI frameworks. We will discuss more practical applications later. Figure 1 (left) shows a screen snapshot of the base system. As seen in this example, the spatial layout reveals distinct group structures. However, these perceptual groups exist only in the user's mind and the computer does not share the structural information. Therefore, the user needs to explicitly select multiple objects in a group to apply some operation to the group. This is particularly frustrating when such group structures change frequently.

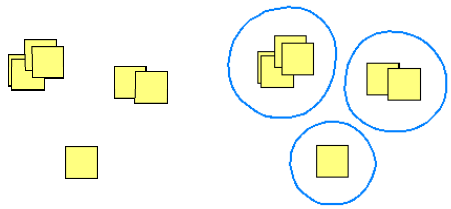

Figure 1: Spatial layout of objects. Left: aggregations of objects represent their group structure. Right: bubbles are visualizations of groups. Each bubble surrounds the objects within a corresponding group.

Our system automatically analyzes a spatial layout and recognizes such implicit structures. We currently focus on the simplest proximity cue for clustering and in future work will support a greater variety of cues [21]. The results of clustering are shown as bubbles surrounding each group of objects (Figure 1, right). A bubble not only works as a visual cue to indicate the existence of a group but also as a handle to interact with the group. The most basic operation is dragging: the user can grab and drag an entire group by dragging the bubble, i.e., the space within the bubble boundary and any objects therein. Other possible operations include deletion and copying of the entire group. These additional commands can be shown in a pop-up menu that appears when the user clicks on the bubble with the right mouse button.

Construction of Bubble Clusters

Bubble clusters are automatically generated by the system according to spatial proximity and there is no explicit

command for creating a cluster. Any operation that changes the layout of objects can be seen as an implicit command to create bubble clusters.

Initially, each object on the canvas is associated with a cluster that consists of a single object. If an object is dragged and approaches another object, their clusters merge (Figure 2(a–c)). As the dragged object moves farther away from the objects in initial proximity, the cluster is automatically split (Figure 2(d–f)). The same is also true when the user drags multiple objects at the same time. Note that the visual representation of bubbles changes during dragging, but the set of objects being dragged does not change during dragging, so newly merged objects do not follow the cursor. If the user wants to move the newly merged objects in the cluster, the user needs to briefly stop and restart dragging, thereby grabbing all of the objects into the new cluster.

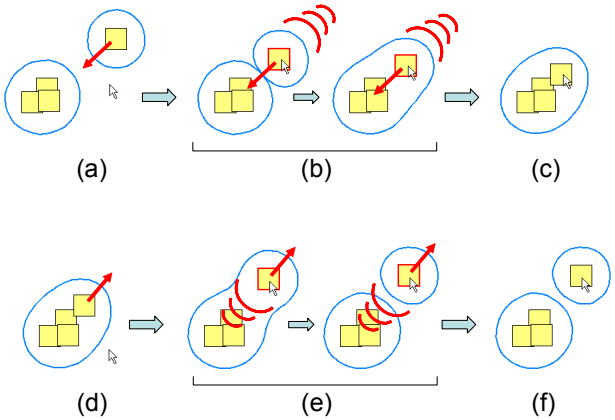

Figure 2: Grouping (a–c) and ungrouping (d–f) of objects.

One interesting operation using this technique is "berry picking." When using a standard interface, to collect multiple scattered objects, a user must repeatedly drag and select the objects. Using bubbles, the user simply repeats drag-release-drag operations to collect the objects, which are automatically added to the cluster when the user briefly releases the cluster near them (Figure 3).

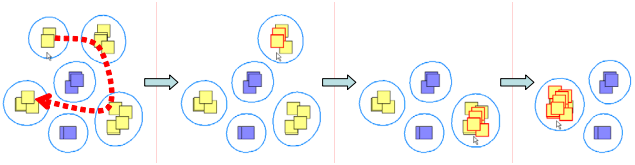

Figure 3: Berry picking.

Some test users of the initial version complained that unnecessary merging of clusters occurred too frequently during dragging in a crowded layout. Therefore, we added a simple hysteresis effect to this merger-and-split process to minimize unwanted occurrences of merging and splitting. Individual clusters resist being merged when approached, and resist being split when objects are leaving. This effect is visualized as bubble surfaces that are pushed away by surrounding bubbles and elongated when an object is leaving (Figure 4). The addition of this small effect greatly increases the flexibility of the spatial layout. For example, two objects at the same distance could belong to the same cluster or not depending on their previous state (Figure 4(b) and (d)).

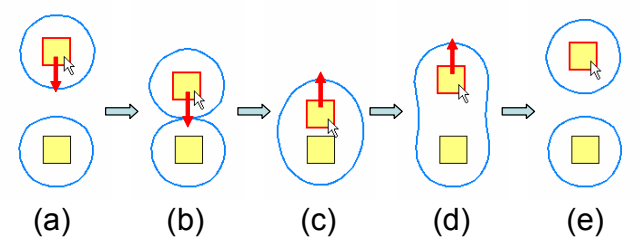

Figure 4: Hysteresis effect on bubbles.

We also provide an explicit cutting operation to split bubble clusters. The user simply draws a free-form stroke across the bubble and the system then splits the original cluster into two separate clusters (Figure 5). Our original intention was to allow the user to select a subset of objects in an existing cluster, but we subsequently found that this cutting operation is particularly useful for recovering the original cluster if the user unintentionally merges two clusters.

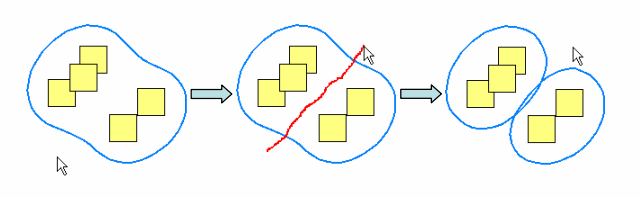

Figure 5: Cutting a bubble. The user draws a stroke and the system splits the cluster.

Spreading a Cluster

Spatial aggregation often involves extensive overlapping of objects, which is necessary to place many objects in a limited space. However, this can cause large overhead in object manipulation tasks because the user needs to manually remove occluding objects one by one in order to see and interact with a hidden object. To help users to interact with such a hidden object, our system supports automatic expansion of a densely aggregated cluster. When the user requests expansion, the system moves the objects in the cluster outward to remove overlapping, while keeping the original configuration as much as possible (Figure 6).

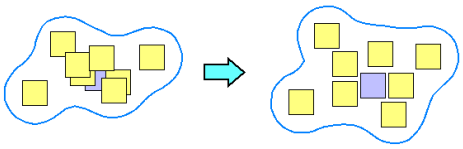

Figure 6: Expansion of a cluster. The user double-clicks a bubble and the system expands the cluster.

Various expansion operations are already in use in some systems. Users can see the contents of folders and piles by opening them [13] in desktop systems. Nonlinear magnification is also frequently used in the context of information

visualization [6]. Ramos et al. proposed a specialized interface for accessing occluded 2D drawings [17]. Our expansion technique is unique in that it solves an optimization problem to preserve the original layout when creating the expanded view. This makes it easier for the user to identify the target object.

Our current implementation uses double clicking on a bubble as the trigger for expansion. The expanded view is temporary; the view immediately returns to the original "closed" view when the user picks a target in the expanded cluster or initiates an operation outside it. Another possible implementation is to leave the expanded cluster "opened" as we did in the user study. It may also be useful to expand a cluster when the cursor pauses on it during dragging to support drag-and-drop into an object hidden in an overlapping aggregation. These specific gestures and behaviors should be customized for each target application.

IMPLEMENTATION

This section describes the implementation details of Bubble Clusters. We first describe how the clustering module constructs a cluster structure for a given object layout, simply by using the distances between objects as criteria for clustering. Then, we describe how the visualization module generates a visual representation of a bubble for each cluster, generating 2D blobby shapes around objects. We also describe how the splitting module processes cutting operations. Finally, we describe how the spreading module resolves overlapping of objects.

Clustering

This module takes the layout of objects as input and groups adjacent objects as a cluster. We use a standard bottom-up clustering algorithm: the system first assigns a cluster to each object and recursively merges adjacent clusters. If the distance between two objects in different clusters is shorter than a given threshold, the two clusters are merged. This merger process is repeated until all adjacent clusters are merged.

In order to add the hysteresis effect, whereby objects resist being connected when approached and resist being separated when left, we use different thresholds for objects depending on whether they belong to the same cluster or to a different cluster in the previous state. If the objects belong to the same cluster, the system uses a large threshold to prevent their disconnection. If the objects belong to different clusters, the system uses a smaller threshold to resist their connection. Pseudo-code is as follows.

```
// c(o) means the cluster containing the object o
threashold[oi,oj] ← small value if c(oi) ≠ c(oj)
                    large value if c(oi) = c(oj)
for ( all objects oi )
        c(oi) = new cluster();
for ( all object pairs oi, oj )
        if ( distance(oi, oj) < threthshold[oi, oj]) )
                merge( c(oi), c(oj) );
```

Our initial implementation identified clusters *after* constructing the visual representation. That is, the system first traced the iso-contours of the potential field around objects and then collected the objects enclosed by each iso-contour as a cluster. This strategy guarantees that the visual representation always matches the internal semantic cluster structure. However, the process of identifying the objects enclosed by each iso-contour is time-consuming. In addition, it is difficult to implement the hysteresis effect in this approach. Therefore, we decided to identify clusters first and generate the visual representation later.

The separation of clustering and visualization computation can cause a mismatch between them; that is, a single cluster can be split into multiple bubbles through the iso-surface extraction. To prevent this, we inserted a post-process that detects when the objects belonging to a cluster would be split in the visual representation. If this happens, the system divides the cluster according to the visual representation and recomputes the bubble geometry.

Visualization

This module takes the set of objects belonging to a cluster and generates a boundary (bubble) for the cluster. We use a 2D version of blobby shapes [4] to do this. Each object is associated with a potential field around it (Figure 7) and the system traces the iso-contour of the potential field in 2D space. If multiple objects exist, the potential fields of nearby objects are summed to generate a smoothly curved boundary. We compute the value of the potential field at each grid point and apply a standard marching-squares method [12] to extract iso-contours.

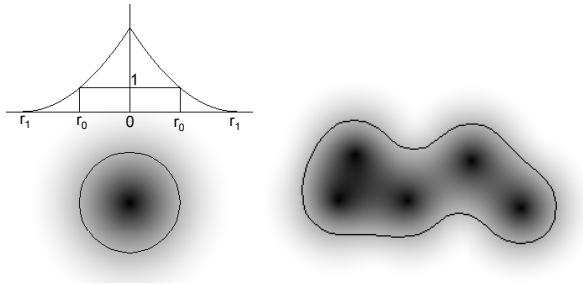

Figure 7: Potential field of a single object. We currently use a radial quadratic function $f(r)$ that satisfies $f(r_0) = 1$, $f(r_1) = f'(r_1)=0$, where r_0 is the bubble radius of an isolated object and r_1 is the size of the field.

We extend the above basic procedure in two ways to obtain more convincing visual effects. First, we attenuate the potential field of densely positioned objects to prevent any unnecessary bulging effect. In the basic algorithm, two overlapping objects result in a larger bubble than that of a single object (Figure 8). However, this is inconvenient, as it consumes unnecessary space and contradicts the clustering algorithm, which uses a predefined distance threshold. We therefore first compute the potential field at each object's location using uniform weights (h_0) and then reduce the weight of each object if the potential at the object's location is large. In our current implementation, we set w =

h_{max} / h_0 where h_{max} is the value of the potential field at the center of an isolated object. This ensures that the weight equals one if there are no other objects within the periphery and decreases as the number of objects within the periphery increases, keeping the bubble size nearly constant.

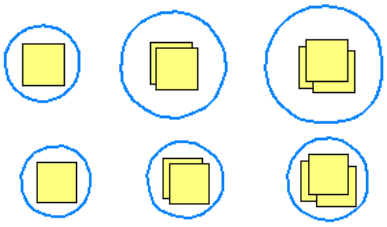

Figure 8: Bubbles with and without bulging. Top: Bubbles created using uniform weights exhibit bulging. Bottom: Bubbles created using adjusted weights do not exhibit bulging.

Second, we add a negative effect to objects outside the cluster to visualize a subtle repulsion effect, i.e., a bubble pushes away surrounding bubbles (Figure 9). When computing the potential of a grid point, the system first visits all objects within the cluster and adds their positive potential values. The system then visits objects outside the cluster and adds their negative potential values. This effect is very subtle. We currently use a weight of −0.4 for the potential value of outside objects.

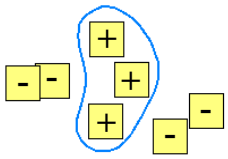

Figure 9: Negative effects of surrounding objects.

Splitting

This module takes a cutting stroke drawn by the user and splits a cluster cut by the stroke. The system identifies the bubble that the stroke intersects and divides the bubble in half at the boundary specified by the stroke. The system then collects the objects inside each half and assigns a new cluster to them. Finally, the system computes the new bubble geometry for each cluster and displays them on the screen. Our current implementation simply ignores incomplete cutting strokes such as those finishing inside a bubble. If the stroke intersects multiple bubbles, the system splits only the first bubble.

Spreading

This module takes overlapping objects in a cluster and returns new positions that avoid overlapping. The simplest approach is to use an iterative repulsion method whereby the system repeatedly identifies overlaps between pairs of objects and pushes them apart. Although very easy to implement, this local repulsion method takes too long because any movement to resolve a local overlap can cause a new overlap in a different area. Therefore, we construct a global system that considers all objects concerned and computes their target locations by solving the global system.

The system first applies Delaunay triangulation to the centers of the objects to find the neighbors of each object. Each edge of the Delaunay triangles corresponds to a pair of objects that are immediate neighbors. For each edge of the resulting triangles, that is, for each pair of neighboring objects, the system computes a target relative position that does not cause overlapping and is close to the original relative position. If two objects are not overlapping, the system uses the current relative position as the target. If they are overlapping, the system scales the relative position vector to resolve overlapping (Figure 10). If two objects have coinciding centers, the system moves them in a random direction.

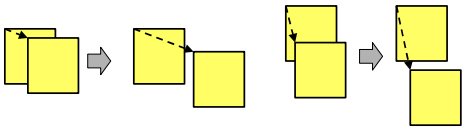

Figure 10: Computation of target relative positions.

Given the target relative positions obtained in the above procedure, the system then computes new absolute positions for the objects so that the resulting relative positions are as close to the target relative positions as possible. We also constrain the center of gravity of all objects to stay at the same location to obtain the absolute positions. Since this is an over-constrained problem in general, we minimize the difference in a least-squares sense as follows:

$$\arg\min_{v}\left\{\sum_{i,j}\left|v_i - v_j - dv_{ij}\right|^2 + \left|\left(\frac{1}{n}\sum_i v_i\right) - c\right|^2\right\}$$

where c is the center of all vertices, dv_{ij} represents the target relative position of v_i seen by v_j, and the summation is taken for all vertex pairs appearing in the Delaunay triangulation. This is a standard least-squares minimization problem for a linear system and can be solved very efficiently using a publicly available sparse linear solver [5]. This approach is similar to the differential mesh processing methods that are frequently used in the computer graphics community [10]. Pseudo-code is as follows.

```
// n ← # of objects; m ← # of edges
for ( edge e_i in the Delaunay triangulation )
        b[i] ← compute target edge vector for e_i
b[m] = center of all objects
A ← (see below)
v = argmin(v){ |Av-b|^2 } = (A^T A)^-1 A^T b;
for (all objects o_i)
        o_i.position ← v[i];
```

where A is a matrix that maps object coordinates to edge vectors and the center of all objects. It looks like

$$\begin{array}{c} \overset{\longleftarrow\ n\ \longrightarrow}{} \\ \left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right\} m+1 \end{array} \begin{pmatrix} 1 & & -1 & & \\ & -1 & & 1 & \\ & & \vdots & & \\ -1 & & & & 1 \\ 1/n & 1/n & \cdots & 1/n & 1/n \end{pmatrix}$$

This method can efficiently resolve existing overlaps in the object layout with a single-pass computation. However,

least-squares minimization can still leave some overlaps, especially when the object layout significantly changes. Therefore, we repeat the above procedure (Delaunay triangulation and least-squares minimization) several times to obtain a satisfactory result.

APPLICATIONS

We believe that Bubble Clusters is widely applicable to the management of transient group structures in many application domains. In this section we discuss some applications of Bubble Clusters and their possible extensions.

Desktop Icons

A virtual desktop is often used as a temporary workspace in which users manage incoming files before filing them into folders. Some users explicitly create lightweight group structures by placing related files together. In an informal investigation of user desktops, for example, we found one user who had placed more than 200 icons on his desktop and created distinct spatial groups (Figure 11). Manipulation of icons in such dense layouts is tedious and Bubble Clusters would make it easier.

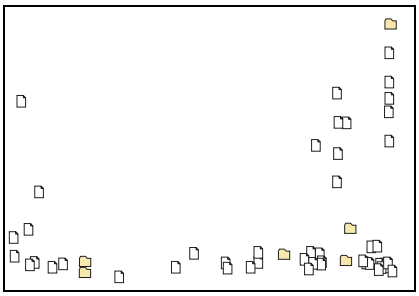

Figure 11: Example of a user's desktop. There are more than 200 overlapping icons on the bottom right corner.

We implemented a prototype desktop using Bubble Clusters to test the feasibility of this idea (Figure 12). Several extensions were made to the original simple Bubble Clusters. First, we implemented a pop-up menu that appears when the user clicks on the bubble with the right mouse button. The menu includes commands for deleting the cluster, beautifying the layout of the objects in the cluster, and transforming the cluster into a folder. We found transformation to a folder to be useful for finalizing a structure after experimenting with many different structures in Bubble Clusters.

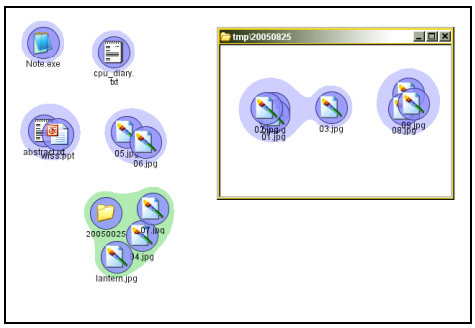

Figure 12: Prototype desktop using Bubble Clusters.

Second, to allow users to drag-and-drop an icon onto a specific icon within a dense aggregation, we added a gestural interface for spreading icons. To drop an icon onto another icon hidden in an aggregation, the user moves the cursor back and forth over the aggregation while dragging. The aggregation then automatically expands to make the target icon visible. A version in which an aggregation expands if the user hovers over it was tested, but the idle waiting time proved frustrating to users.

Web Page Bookmarks

Bookmarks are a good example of rapidly changing personal information. Users' interest in bookmarked pages and the content of those pages change frequently. Consequently, new bookmarks are often added and old bookmarks quickly become obsolete. Accordingly, the group structures of bookmarks need to change rapidly. However, it is tedious to work with a tree structure, and most bookmark lists remain messy.

The Data Mountain system [18] proposes the use of a spatial layout for managing bookmarks. The user can place relevant bookmarks together in the workspace, and is freed from inflexible hierarchical structures. We believe that Bubble Clusters can further facilitate the use of spatial layouts for managing bookmarks. Bubble Clusters would help users to move sets of bookmarks easily and to find bookmarks hidden in dense aggregations.

Playlists for Music Players

A playlist is a collection of songs in a music player, and sets of playlists are used to organize the large numbers of songs collected by users. Playlists are also shared and exchanged, and are therefore another example of a constantly changing group structure. We believe that Bubble Clusters can help in their organization. Goto et al. proposed a music playback interface for managing large numbers of tunes [7] where each song is represented as a small circular icon, and the user organizes playlists by manipulating them. Bubble Clusters can be a natural extension of this kind of interface.

Digital Inking

Handwritten text on a blank canvas, as in free-form note-taking systems and electronic whiteboard systems, is another good application for Bubble Clusters. The strokes in such text form distinct group structures and users frequently move or delete sets of strokes as a group; users seldom move a single stroke in a word. However, most systems require the use of lasso selection or rubber banding to select target strokes, which interferes with the users' fluent interaction with the system. Some systems automatically recognize implicit structures in handwritten text [14, 19] but they require explicit requests from the user. Bubble Clusters can work in the background and provide a way to manipulate groups of strokes without distracting from the main writing task. Our interface can complement existing informal grouping methods for handwritten texts [11, 15]. These are designed for managing larger working units (e.g., the entire discussion on a single topic) and our method is effective for smaller units (e.g., words and sentences).

We have implemented a prototype Bubble Clusters interface for free-form inking (Figure 13). This demonstration was well received by lecturers using electronic whiteboards, many of whom would end up dragging a single stroke when they attempted to drag an entire word. Computing the potential fields for free-form strokes is computationally expensive because the strokes consist of many points while a single point is sufficient for an icon. Therefore, we decided to update the bubble geometry only when the user completes a stroke or finishes moving strokes.

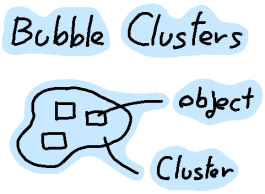

Figure 13: Prototype interface for free-form inking.

USER STUDY

We conducted two user studies to measure the performance of our technique in different application scenarios: a grouping task in a desktop environment and an ink relocation task in a digital whiteboard environment. The goal of these studies is to see whether the users can successfully understand the behavior of bubble clusters and to see whether automatic grouping actually improves user performance.

User Study 1: Grouping Icons

The goal of the first study was to examine whether Bubble Clusters could help users with icon grouping in a desktop environment. The participants were asked to group icons with same color using conventional folders and Bubble Clusters. This kind of grouping task occurs when the user wants to organize a set of unclassified files, such as images and documents. Our main hypothesis was that the Bubble Clusters would outperform conventional folders in this kind of bottom-up grouping task.

Apparatus. We used a Dell Dimension 9100 Pentium 4 computer with a standard mouse and keyboard. The test program was implemented using Java™ 5.0 running on Windows XP SP2 Professional. The size of the canvas was 1024 × 736 pixels.

Participants. Twelve participants, all men ranging in age from 20 to 25 years, were recruited mainly from the local university community for the experiment. Participants each were paid 1,000 Japanese yen and all were regular computer users. None had previous experience with Bubble Clusters.

Task. The task was to group spatially distributed icons according to their colors. Each task consisted of two phases: an initial grouping phase and a regrouping phase. Initially, small boxes were scattered randomly on the canvas (Figure 15(a)). Each box was colored red, green, blue, cyan, or yellow. Users were given 6 boxes of each color, for a total of 30 boxes. Users grouped the scattered boxes by color using dragging operations (initial grouping task). The grouping task was complete when all of the boxes were correctly grouped (Figure 15(b)), and then the color of each box was randomly changed (Figure 15(c)). The number of colors and the number of boxes per color were the same as in the initial grouping task. The users then grouped the boxes again (regrouping task; Figure 15(d)).

Interfaces. In the *folder* interface, the users distributed the colored boxes into folders (Figure 14). This task mimics the standard desktop workspace in a modern window system. The users could create a new folder or change its name from a context menu. The users could open a folder by double-clicking. The users moved boxes into a folder by dragging them into the folder or into its corresponding open window. Multiple boxes could be selected with standard rubber banding. We also provided additive selection using a shift key. Each task ended when all of the boxes were correctly grouped, with all of the boxes of the same color in the same folder and each folder containing only one color of box, although we allowed the users to leave folders empty.

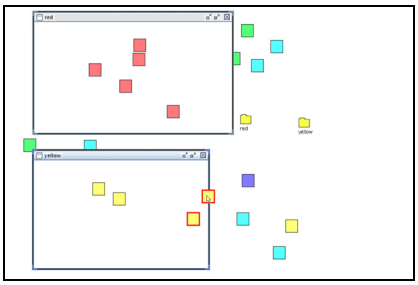

Figure 14: Screen snapshot of the grouping task using the *folder* interface.

In the *bubble* interface, the participants used Bubble Clusters to create clusters with boxes of the same color (Figure 15). Users could move, merge, split, and expand bubbles as described in this paper. Each task ended when all of the boxes were correctly clustered, with each cluster containing all of the boxes of the same color.

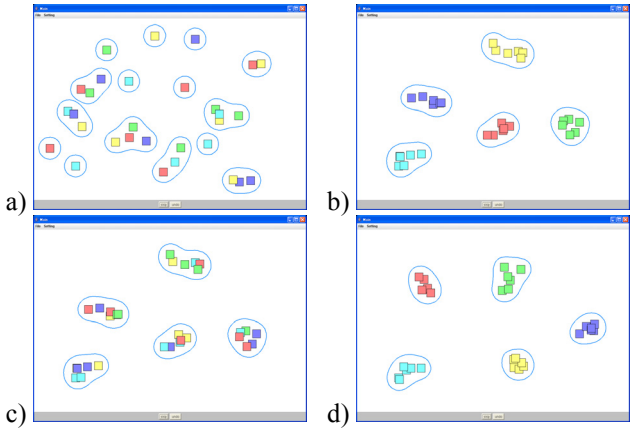

Figure 15: Screen snapshots of the icon grouping task using the *bubble* interface. (a) Initial layout. (b) Completion of the initial grouping task. (c) Colors are changed. (d) Completion of the regrouping task.

The major difference between the two interface conditions is that the folder interface requires explicit creation and opening of folders. This can be seen as a bias. However, the ability to create and work with groups without explicit group operations is the main benefit of our technique. Therefore, we set the goal of this study to show the benefit of removing explicit group operations and chose this design.

Design. Each participant was subjected to both interface conditions. Participants were divided into two groups of six. One group used the *folder* interface first and the other group used the *bubble* interfaces first. Each session lasted approximately 20 min.

One set consisted of the grouping and regrouping phase, and each participant performed 4 sets for each interface, for a total of 16 trials (4 sets × 2 interfaces × 2 phases). The first set in each interface was used for tutorial and practice. The subsequent three sets were used for the statistical analysis. Participants were asked to complete the trials as quickly and as accurately as possible. The task time was measured from the moment the user pressed a start button until the moment when all of the boxes were grouped or regrouped. The users were requested to resolve errors before completing the task, which means that error measurement is included in the task completion time. Participants completed a questionnaire at the end of the experiment.

Results. The average and the standard variation of the two interfaces are shown in Figure 16. Paired t-tests showed significant differences between the two interfaces in both the grouping tasks ($t(11) = 6.46$, $p < 0.001$) and the regrouping tasks ($t(11) = 9.61$, $p < 0.001$). In the questionnaires, 10 of 12 participants preferred the *bubble* interface over the *folder* interface for this particular task and all of the participants answered "yes" to the question "Was the concept of Bubble Clusters easy to understand?" This shows that Bubble Clusters can be useful for bottom-up grouping of items on the desktop.

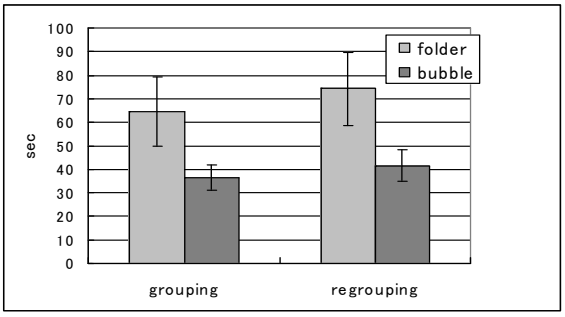

Figure 16: Average trial time in seconds under each interface condition with standard deviation.

A detailed analysis of the user's operation history under the bubble condition reveals that the participants successfully understood each operation and used it well to accomplish the task. On average, the participants used 8 cut operations and 6 expansion operations in the grouping phase, and 10 cuts and 12 expansions in the regrouping phase. This difference occurred because the icons were highly overlapping at the beginning of the regrouping phase and the user first needed to resolve this before the task could be completed.

User Study 2: Ink Relocation

The goal of the second study was to examine whether Bubble Clusters could help users perform an ink manipulation task in a digital whiteboard environment. The participants were asked to move handwritten words to designated areas using conventional lasso selection and Bubble Clusters. Our main hypothesis was that the Bubble Clusters would outperform conventional lasso selection.

Apparatus. We used a display-integrated tablet with a 17" display (Wacom Cintiq) connected to a laptop computer with 1.10 Ghz Pentium M processor (Toshiba Dynabook SS2120). The test program was implemented using Java™ 5.0 running on Windows XP SP2 Home Edition. The size of the screen was 1024 × 768 pixels.

Participants. Twelve participants (2 women and 10 men) ranging in age from 18 to 35 years were recruited mainly from the local university community and volunteered for the experiment. Participants each were paid 1,000 Japanese yen and all were regular computer users. None had previous experience with ink manipulation using Bubble Clusters. All participants were right-handed.

Task. The task was to move eight handwritten words, listed vertically on the left-hand side of the screen, to the right-hand side in a reversed order. Each word consisted of multiple free-form strokes. The target region for each word was clearly shown to the user. Figure 17 shows a screen snapshot of task using the lasso interface.

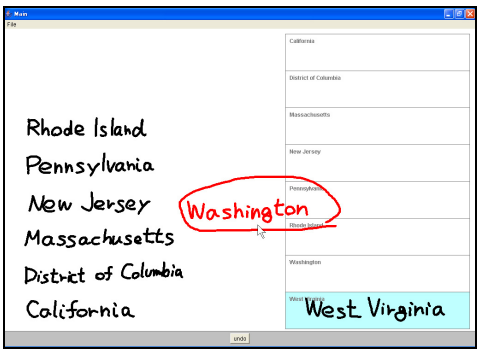

Figure 17: Screen snapshot of the ink relocation task using the lasso interface.

Interfaces. We tested four interface conditions. The *Lasso* interface used standard lasso selection in which the user selected a target word by drawing an enclosing stroke around it and moved the word by dragging the area enclosed by the lasso (Figure 17). For Bubble Clusters, we tested three different parameter settings. The *small bubble* interface used Bubble Clusters that were too small to enclose some words; these words were split among multiple bubbles (Figure 18, left). In this case, the user needed to connect these bubbles to move the entire word. In the *medium bubble* interface, bubble size was optimal and each word was associated with a single bubble (Figure 18, cen-

ter). The user could simply drag-and-drop the individual bubbles to complete the task. In the *large bubble* interface, all of the words initially were connected together (Figure 18, right). The user had to cut the bubbles to move the individual words. These scenarios emulate errors in automatic group creation in real-time authoring.

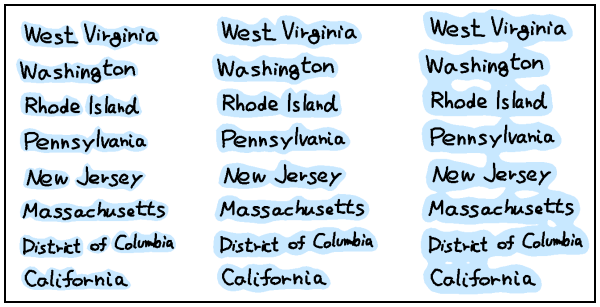

Figure 18: Three different bubble conditions (left, small bubble; center, medium bubble; right, large bubble).

Design. A balanced within-subjects design was used. Each participant was subjected to all four interface conditions. Participants were divided into two groups of six. One group used the lasso interface first and the other group used the bubble interfaces first. The order of three bubble conditions was counterbalanced within each group. Each session lasted approximately 20 min.

Each participant performed 5 trials under each interface condition, for a total of 20 trials (5 trials × 4 interfaces). The first two trials were used for tutorial and practice and the subsequent three trials were applied in the statistical analysis. Participants were asked to complete the trials as quickly and as accurately as possible. Pilot testing and analysis of the study data did not show significant learning effects after the first two trials. The task time was counted from the moment the user pressed a start button until the moment when the last word was placed within the target region. Participants completed a questionnaire at the end of the experiment.

Results. The average and the standard variation of the four interfaces are shown in Figure 19. A repeated measures analysis of variance (ANOVA) was conducted on the task completion times across the four interface types (lasso, small bubble, medium bubble, and large bubble). A significant difference was observed across the interface types, ($F(3, 33) = 31.651$, $p < 0.001$). The mean task completion time was 32.72 s for lasso selection, 24.73 s for small bubble, 19.28 s for medium bubble, and 28.86 s for large bubble. Post hoc tests showed that tasks using the small bubble and the medium bubble took significantly less time than the lasso selection ($p < 0.01$ and $p < 0.001$, respectively), but we found no significant difference between using the lasso selection and the large bubbles ($p > 0.1$).

These results show that Bubble Clusters can be very useful for an ink relocation task. Under the optimal condition (medium bubble), the task completion time was approximately 60% that under the lasso condition. The advantage of Bubble Clusters remained significant when the bubble size was too small and the user needed to perform an extra operation. In a realistic situation, most bubbles would be of an appropriate size (because the user can easily customize bubble size) with a few incorrectly clustered words. Therefore, the real performance should be within the range of the three bubble conditions in this study, which is clearly faster than the lasso condition.

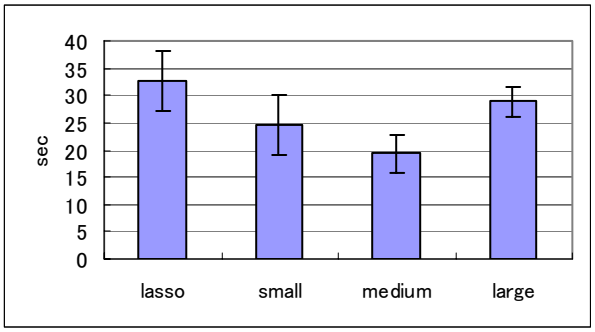

Figure 19: Average trial time in seconds under each interface condition with standard deviation.

The observation of the users' behavior during the study and the posttest interview revealed that the large bubble condition has a usability issue. Several users unintentionally dragged a large bubble when they tried to cut the bubble. It might be possible to reduce this mistake with a more complicated cutting interface, such as one that uses a physical button or a special gesture, but these are not attractive options because they would make the interface more complex. The small bubble condition had better results than the large bubble condition, so a practical solution to this problem is to use bubbles that are slightly small (i.e., use a parameter setting that tolerates over-segmentation while avoiding under-segmentation).

CONCLUSIONS AND FUTURE WORK

We proposed a technique, Bubble Clusters, for organizing transient group structures in spatial information management systems. Nearby objects are automatically recognized as a cluster and the system visualizes the structure as a bubble surrounding the objects. We also introduced automatic expansion of overlapping objects, implemented a prototype system, and performed a user study. The results show that Bubble Clusters could improve performance in a simple icon grouping task and an ink relocation task compared to standard folders and lasso selection, respectively. Although they do not guarantee the superiority of our technique in real applications, these results show that Bubble Clusters did help users in certain situations.

We hope to further explore several additional areas. Currently, users can select an icon or a bubble, but there is no method to select a smaller group of icons within a bubble. It might be interesting to vary the selection range depending on the number of clicks. For example, a single-click might select the icon underneath the cursor, a double-click, the overlapping icons within a cluster, and three clicks, the entire cluster [9]. Such multiple-click selection is also seen

in text editors. Another possibility is to change the selection range depending on where the user clicks in a bubble. That is, clicking near an icon could select only nearby icons while clicking near the edge of a bubble selects the entire bubble.

Scalability can be an issue when users want to manage a large number of objects using Bubble Clusters. Our initial goal was to improve the usability of existing spatial layout systems, so the target number of objects to handle was less than a hundred. Beyond that, we expect users to switch to traditional folders. One interesting future direction is to combine Bubble Clusters with zooming interfaces to obtain a virtually infinite amount of space [3].

Visual clutter is also an issue. Although we designed the visual representation to be as simple as possible, it does show a great deal of information on the screen and can be distracting. One solution is to make bubbles transparent and only show them at relevant times in the interaction. In the electronic whiteboard application, it would be helpful to show clusters only on the instructors display and hide them on the main display.

Some test users said that they wanted to name clusters. Currently, users cannot name clusters manually because the system creates clusters automatically. The users requested automatic naming of clusters so that they had some clue as to the kind of icons hidden in a dense cluster. It might be possible to add properties to a cluster based on the properties of the objects in the cluster. One possibility is to give a different color to each bubble depending on the properties of its member objects. These extensions would allow users to recognize the characteristics of a cluster without examining individual objects.

REFERENCES

1. Agarawala, A. and Balakrishnan, R. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of CHI*, pp. 1283-1292, 2006.
2. Bauer, D., Fastrez, P., and Hollan, J. Spatial tools for managing personal information collections. In *Proceedings of HICSS'*, pp. 104.2, 2005.
3. Bederson, B.B., Hollan, J.D., Perlin, K., Meyer, J., Bacon, D., and Furnas, G. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing, Volume 7, Number 1*, pp. 3–31, 1996.
4. Blinn, J.F. A generalization of algebraic surface drawing. In *ACM Transactions on Graphics, Volume 1, Issue 3 (July 1982)*, pp. 235–256, 1982.
5. Davis, T.A. *UMFPACK Version 4.1 User Guide*, 2003.
6. Furnas, G.W. The fisheye view: a new look at structured files. Technical Report, Bell Laboratories, 1981.
7. Goto, M. and Goto, T. Musicream: new music playback interface for streaming, sticking, sorting, and recalling musical pieces. In *Proceedings of the 6th International Conference on Music Information Retrieval*, pp. 404–411, 2005.
8. Grossman, T. and Balakrishnan, R. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proceedings of CHI*, pp. 281–290, 2005.
9. Igarashi, T., Matsuoka, S., and Masui, T. Adaptive recognition of implicit structures in human organized layouts. In *Proceedings of Visual Languages '95*, pp. 258–266, 1995.
10. Igarashi, T., Moscovich, T., and Hughes, J.F. As-rigid-as-possible shape manipulation. In *ACM Transactions on Computer Graphics, Volume 24, Issue 3, ACM SIGGRAPH 2005*, pp. 1134–1141, 2005.
11. Kramer, A. Translucent patches—dissolving windows. In *Proceedings of UIST*, pp. 121–130, 1994.
12. Lorensen, W.E. and Cline, H.E. Marching cubes: a high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH*, pp. 163–169, 1987.
13. Mander, R., Salomon, G., and Wong, Y.Y. A 'pile' metaphor for supporting casual organization of information. In *Proceedings of CHI*, pp. 627–634, 1992.
14. Moran, T.P., Chiu, P., van Melle, W., and Kurtenbach, G. Implicit structure for pen-based systems within a freeform interaction paradigm. In *Proceedings of CHI*, pp. 487–494, 1995.
15. Mynatt, E.D., Igarashi, T., Edwards, W.K., and LaMarca, A. Flatland: new dimensions in office whiteboards. In *Proceedings of CHI*, pp. 346–353, 1999.
16. Perlin, K. and Fox, D. Pad: an alternative approach to the computer interface. In *Proceedings of SIGGRAPH*, pp. 57–64, 1993.
17. Ramos, G., Robertson, G., Czerwinski, M., Tan, D., Baudisch, P., Hinckley, K., Agrawala, M. Tumble! Splat! Helping users access and manipulate occluded content in 2D drawings. In *Proceedings of AVI*, pp. 428–435, 2006.
18. Robertson, G., Czerwinski, M., Larson, K., Robbins, D.C., Thiel, D., and van Dantzich, M. Data Mountain: using spatial memory for document management. In *Proceedings of UIST*, pp. 153–162, 1998.
19. Saund, E., Fleet, D., Larner, D., and Mahoney, J. Perceptually-supported image editing of text and graphics. In *Proceedings of UIST*, pp. 183–192, 2003.
20. Saund, E. and Moran, T.P. A perceptually supported sketch editor. In *Proceedings of UIST*, pp. 175–184, 1994.
21. Shipman, F.M., III, Marshall, C.C., and Moran, T.P. Finding and using implicit structure in human-organized spatial layouts of information. In *Proceedings of CHI 1995*, pp. 346–353, 1995.
22. Worden, A., Walker, N., Bharat, K., and Hudson, S. Making computers easier for older adults to use: area cursors and sticky icons. In *Proceedings of CHI*, pp. 266–271, 1997