PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps

Benjamin B. Bederson
Human-Computer Interaction Laboratory
Computer Science Department, Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
+1 301 405-2764
bederson@cs.umd.edu
http://www.cs.umd.edu/hcil/photomesa

ABSTRACT
PhotoMesa is a zoomable image browser that uses a novel treemap algorithm to present large numbers of images grouped by directory, or other available metadata. It uses a new interaction technique for zoomable user interfaces designed for novices and family use that makes it straightforward to navigate through the space of images, and impossible to get lost.

PhotoMesa groups images using one of two new algorithms that lay out groups of objects in a 2D space-filling manner. *Quantum treemaps* are designed for laying out images or other objects of indivisible (quantum) size. They are a variation on existing treemap algorithms in that they guarantee that every generated rectangle will have a width and height that are an integral multiple of an input object size. *Bubblemaps* also fill space with groups of quantum-sized objects, but generate non-rectangular blobs, and utilize space more efficiently.

Keywords
Zoomable User Interfaces (ZUIs), Treemaps, Image Browsers, Animation, Graphics, Jazz.

INTRODUCTION
There has been much work in recent years on information retrieval systems for multimedia, including systems concentrating on images. However, these systems focus on specifying queries or presenting results in a manner that helps users quickly find an item of interest. For image searches, in particular, there has been relatively little work on new interfaces, visualizations, and interaction techniques that support users in browsing images.

Image browsing is important for a number of reasons. First of all, no matter what information retrieval system is being used, the user has to browse the results of the search. It is certainly important to build query systems that help users get results that are as close to what is wanted as possible. But there will always be images that need to be browsed visually to make the final pick.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
UIST 01 Orlando FLA
Copyright ACM 2001 1-58113-438 -x/01/11...$5.00

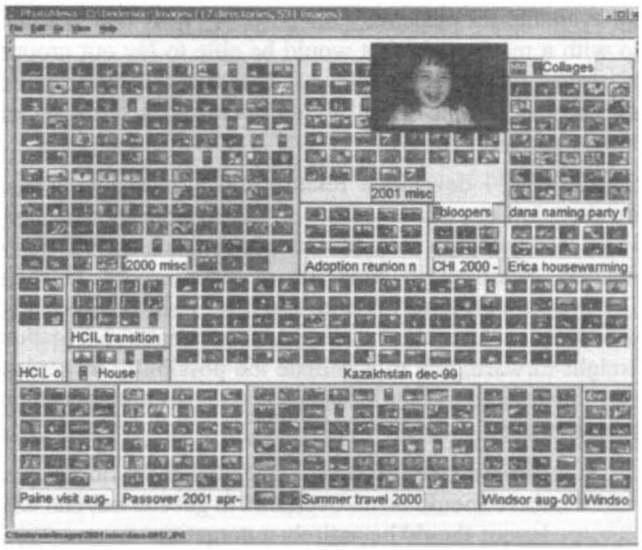

Figure 1: Screen snapshot of PhotoMesa with over 500 images in 17 groups.

Most image browsing systems present the images as a grid of thumbnails that the user can scroll through with a vertical scrollbar, and see a high resolution version of the image with some mouse interaction. There are also a few alternative designs, such as manually constructed digital photo albums, and one commercial zoomable image browser.

A second reason for needing new image browsers is more subtle, and was actually my primary motivation for doing the present work. Sometimes, people browse images just for the pleasure of looking at those images, and they often do it with other people. This is especially true for personal photos. As people take more digital family pictures, we need better tools to support users in home settings as they look at those pictures together on a computer screen. Looking at home photos has a lot of overlap with traditional retrieval systems. People still want to be able to find photos of particular people and events, etc. However, they are less likely to be time pressured to find a particular photo, and more likely to be interested in serendipity – that is, finding photos they weren't looking for [6].

I found I needed better tools to look at pictures with my two-year-old daughter. I did not want to spend the time to make custom "albums". In addition, I found using traditional software with a grid of thumbnails, scrollbars, and popup viewer windows unpleasant in this context. I wanted to concentrate on the images – and more importantly, as I was looking at the photos with my daughter, it was crucial that she be an active part of the interaction, and not just a passive bystander.

Motivated by the need of a tool that would support browsing of images with my family, I started to investigate techniques for presenting collections of images or other visual data. While much work has been done on visualizing complex datasets, surprisingly few techniques are available for presenting images. My goal was to come up with a mechanism that would be able to lay out groups of images automatically in a way that would offer a simple interface to browse while giving access to a large set of images and their context.

To this end, I developed PhotoMesa, a zoomable image browser that organizes images in a two-dimensional grid, where images with a shared attribute (such as directory location, nearness in time, or a shared word in their filename) are grouped together (Figure 1). It uses zooming and simple interaction techniques to make navigation straight-forward, and to eliminate the possibility of getting lost. In building PhotoMesa, I kept the following design goals in mind:

- Simple to use (interaction should focus on images, there should be no overhead to get started, and any layout should be entirely automatic)
- Work well for family-use settings, encouraging shared co-present use
- Support collections of photos, and use screen space efficiently

To lay out the groups of images automatically, I ended up developing two new algorithms, called *quantum treemaps* and *bubblemaps*. Quantum treemaps are a variation on existing treemap algorithms [21]. Treemaps are a family of algorithms that partition two-dimensional space into regions that have an area proportional to a list of requested areas. The problem with existing treemap algorithms is that they return areas of arbitrary aspect ratios. A requirement of photo display is that the regions that show groups of photos must have dimensions that are integer multiples of the dimensions of the photos – that is, they must be sized to contain *quantum*, or indivisible contents. The use of treemaps to display images is the first known use of treemaps to display visual content, such as images, rather then just using the size and color of the rectangles to visualize two numerical attributes of a dataset.

The bubblemap algorithm generates non-rectangular groups. The groups are generated with a grid-based recursive fill algorithm. They fill all the cells in a grid leaving almost no unused space, and generate groups of images that are approximately rectangular or circular.

This paper describes PhotoMesa and the quantum treemap and bubblemap layout algorithms. All the software described in this paper is written in Java 2, is fully functioning as described, and is available at http://www.cs.umd.edu/hcil/photomesa.

RELATED WORK

As mentioned previously, the standard way to let users browse a set of images is with a grid of thumbnails with a vertical scrollbar. Clicking on an image thumbnail usually brings up a window with the high-resolution version of the image. The user then has to manage the open windows manually, and close them when they are no longer needed. One good commercial example of this approach is ACDSee which offers a clean interface and fast interaction [1].

This approach has been extended by a research group at the University of Maryland developing PhotoFinder [16, 22]. It lets users organize photos into "collections" which are displayed with a representative image that the user selects. The interface first shows collections, and selecting a collection displays a traditional grid of thumbnails. PhotoFinder avoids the problem of window management, by displaying high-resolution photos in a pane within the interface. The PhotoFinder project concentrates on interfaces for managing and searching a database of meta information, but the browsing interface is essentially a polished traditional approach.

Document Lens is a technique that uses 2D fisheye distortion to present a grid of thumbnails of documents with a mechanism to zoom one document up to a readable size in place [18]. Document Lens, however, presents just a single collection of objects at a time.

Others have looked into automated algorithms for clustering semantically related information, and presenting the results visually. Hascoët-Zizi and Pediotakis built such a system for a digital library retrieval system, showing the available thesaurus as well as results of searches [14]. Platt has built a system for automatically clustering photos, and extracting representative photos for each cluster [17].

Several groups have investigated applications of images for story telling or sharing in the home. The Personal Digital Historian project at MERL is building a circular display on a tabletop intended for several people to interact with images together. The design includes search by several kinds of metadata, but the mechanism for interacting with many images was not described in detail [20]. This is an example of support for co-present use which is a theme described in some of the author's prior work [23].

A group at Ricoh is building a dedicated portable story-telling device based on the construction of sequences of images. It has a dedicated hardware interface for selecting sequences of images which can then be annotated with audio, and played back when telling the story associated with those images [6].

For a pure software approach, we and others have built Zoomable User Interfaces (ZUIs) for image browsing. ZUIs are interfaces that present information on a large flat space where the user can smoothly zoom into the space to see information in more detail, or zoom out to get an overview. ZUIs have the potential advantages that they are easy to comprehend, and they give a consistent and easy to use interface for getting an overview of the information, and seeing more detail.

An earlier ZUI-based image browser was ZIB (Zoomable Image Browser) [11]. ZIB combined a zoomable presentation of a grid of images with a search engine (that searched metadata), and a history mechanism to access previous searches. However, ZIB provided access to only a single group of images, and used manual zooming which was difficult to use.

The approach started in ZIB was continued in a new project that is creating an interface for elementary school-aged children to find multimedia information in a digital library [12]. This project, called SearchKids, presents visual results in a zoomable interface with a simpler interaction mechanism that PhotoMesa is based on.

Another ZUI-based image browser is currently available commercially by Canon, and is called ZoomBrowser EX [2]. The Canon browser presents a hierarchy of images (either manually constructed, or imported from a disk hierarchy) with containment. The top level shows a grid of squares, each of which contain a grid of image thumbnails and/or smaller squares that show more thumbnails, etc. It uses a layout very similar to what we used earlier in the Pad++ directory browser [8]. This layout has the disadvantage that all directories are the same size, and the contents are scaled to fit so that images in large directories are scaled small so as to be unreadable.

The interaction is to click on a square, and the contents of the square are smoothly zoomed into. Clicking on an image brings up a traditional high-resolution image viewer in a separate window. Clicking on a special zoom-out button zooms out to the next level in the hierarchy. There is also a magnification mode which zooms in a fixed amount each click, rather than zooming into the next level of the hierarchy.

PHOTOMESA

PhotoMesa allows the user to view multiple directories of images in a zoomable environment, and uses a set of simple navigation mechanisms to move through the space of images. It also supports clustering of images by metadata available from the file system. It requires only a set of images on disk, and does not require the user to add any metadata, or manipulate the images at all before browsing, thus making it easy to get started with existing images.

PhotoMesa is written entirely in Java 2, and is built using the Jazz framework for Zoomable User Interfaces [9]. The name *PhotoMesa* derives from the Spanish word *mesa* which means table, but is commonly used in the US southwestern states to describe the natural volcanic plateaus which are high and have flat tops. Standing atop a mesa, you can see the entire valley below, much as you can see an overview of many photos in PhotoMesa.

To start using PhotoMesa, a user opens a directory, or a set of directories, and PhotoMesa lays out the directories of images in a space-filling manner as shown in Figure 1, using a quantum treemap to create one rectangular group for each directory. Even though a hierarchical directory structure is read in, the images are displayed in a flattened, non-hierarchical manner. The rationale for this is that users looking at images are primarily interested in groups of photos, not at the structure of the groups. In addition, the interface for presenting and managing hierarchies of groups would become more complicated, and simplicity was one of the goals of the PhotoMesa. However, this is a design characteristic of PhotoMesa, not of the of the treemap algorithms which can be applied hierarchically.

As the user moves the mouse, the group the mouse is over is highlighted, and the label is shown in full (it may have been clipped if there wasn't room for it). Then when the user clicks, the view is smoothly zoomed in to that group. Now, a highlight showing a set of images under the mouse lets the user know which images will be focused on when the mouse is clicked again. The number of images highlighted is chosen to be enough to fill about half of the screen so that the user will be able to drill down quickly to a full-resolution single image. At any point, the user can press the right button (or Enter key) to zoom out to the previous magnification. In addition, the user can double-click on an image to zoom all the way into that image and avoid intermediate zoom levels, or the user can double-right click to zoom all the way out to the top level.

The user can also press alt-left/right arrows to move back and forth in their history of views. Or, they can press the arrow keys to pan up, down, left or right. When zoomed all the way into a full-resolution image, the arrow keys stay within the current group of images, wrapping as necessary. When zoomed out so more than one image is visible, the arrow keys move across groups to let the user explore the entire space.

At all times, if the cursor is left to dwell over an image thumbnail for a short time, that thumbnail is zoomed up until it is 200 pixels wide overlaying the other, unchanged images (Figure 1). This preview is immediately removed whenever the mouse is moved.

While it is not necessary for users to do any authoring to browse images with PhotoMesa, they are allowed to change the color of image groups (although group background colors are assigned by default). This can make it easier to make sense of the large display of images since the colored areas can act as landmarks which are known to be effective navigation aids [15].

PhotoMesa supports drag-and-drop to let users directly export images to email, or other applications. Since

emailing photos is a significant use, PhotoMesa automatically reduces the resolution and quality of images when they are dragged out of PhotoMesa. This resolution reduction is controllable through a preference panel. This eliminates the need to go through a special processing step when emailing images.

While the support of browsing is the primary goal of PhotoMesa, it is also sometimes desirable to find images in a specific group, and it can be difficult to scan labels in a 2D space. So, a search pane is available that shows all the directories in order. Mousing over a label highlights the corresponding group of images, and clicking on a label zooms into that group. In addition, the search pane has a search box where users can search for images by words in their filename.

After PhotoMesa was built, and we started using it to browse directories of images, I realized that another way of thinking about what PhotoMesa was doing was presenting a large set of images clustered by directory. So I then added support for clustering by other data. Since I didn't want to require users to add metadata, PhotoMesa uses whatever data is already available in the file system, which is just file date and name. If a user selects view by year, PhotoMesa uses the file date to group all the currently opened photos by year, and creates a layout with one region per year. It does the same thing for viewing by month.

Another clustering technique takes advantage of the fact that people sometimes give meaningful filenames to their images, often with several words per image to describe the contents of the image (Figure 2). If a user selects view by "filename words", it parses the filenames of all of the open images, and creates one cluster for each unique word in a filename (as tokenized with all the standard delimiters and where filename extensions and numeric tokens are ignored). Thus, if an image has 3 words in its filename (such as "ben-eats-cake"), then that image will appear in 3 clusters (one for "ben", one for "eats", and one for "cake").

PhotoMesa computes multiple sized thumbnails for each image, and dynamically loads the appropriate one. In this manner, it maintains good performance, even with large numbers of images. The thumbnails are created the first time an image is loaded, and cached in a special directory managed by PhotoMesa.

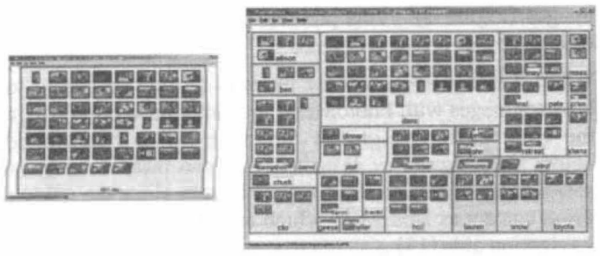

Figure 2: A directory of images (left), and the same images grouped by filename words (right).

The design of PhotoMesa presents an inherent difference compared to traditional scrolling thumbnail grids. The traditional approach has the advantage that it is searchable by navigating in one dimension (through vertical scrolling), while PhotoMesa requires navigation in two dimensions, which is typically harder for users. However, PhotoMesa has the advantage that the user can easily get an overview by zooming out. Through this interaction, the user can control the trade-off between the number of images shown and their resolution. This difference is a direct effect of the zooming nature of PhotoMesa. If a vertically oriented grid of thumbnails were zoomed out, the space would be mostly unused on either side of the linear list, and the display space would thus be largely wasted. Thus, it seems that a 2D zoomable interface and 1D displays of data are inherently incompatible.

USE OF PHOTOMESA

I have used PhotoMesa regularly with my two year old daughter for several months. We load in all of our family pictures (Figure 1) and sit together in front of a laptop computer. She will point at an area and I click and zoom in to it. I keep zooming in as she points at areas until we get all the way in to a single photo. I then zoom out one level, and if she asks to see another photo, I zoom into it. Otherwise, I zoom out another level until she sees something she is interested in. In this fashion, we look at the photos together, and she is able to stay in control and maintains a high level of interest. The zooming and smooth animation make it so that she is clearly able to follow what is going on, even though I operate the mouse.

In addition, over 9,000 people have downloaded PhotoMesa from the web. While this is obviously not a controlled study, it has been informative nevertheless. I have received very positive feedback, sometimes describing use scenarios I did not originally envision. One designer used it as a "disk mapper" to find out what was on her disk. Another put the software with photos on a CD and mailed it to family and friends. Others have envisioned embedding it in a range of applications, from supporting hobbyist aquarium logging to web-based photo sharing. Perhaps most importantly, several people reported they find it ideal to use with their families – supporting my original design goal.

QUANTUM TREEMAPS

In the course of developing PhotoMesa, I ran into a significant problem. I needed an automatic way to lay out groups of images in a visually simple manner that filled all the available space. I started to solve this by looking into treemap algorithms. Treemaps are a family of algorithms that are space-filling partitions of a two-dimensional area. Treemaps take as input, a list of n numbers and a rectangle. They partition the area into n rectangles, one per input number. The rectangles are guaranteed to fill the input rectangle, and each rectangle is proportional in area to a number on the input list. Treemaps are designed to be applied hierarchically, so any given resulting rectangle can itself contain a treemap, and so on, recursively.

In order to build PhotoMesa, I had to extend the treemap algorithms to accommodate fixed size images. To understand this, let us start by looking at existing treemap algorithms.

There are two desirable properties that treemap algorithms can have: generated rectangles with aspect ratios close to 1 (i.e., rectangles that are close to squares), and order.

Here, and for the rest of the paper, aspect ratio is defined as `max((width / height), (height / width))`, so that an aspect ratio of 1 is perfectly square, and aspect ratios larger than one are more rectangular. Rectangles with aspect ratios close to 1 are desirable because, generally speaking, they are more visually attractive. In addition, humans seem to be able to estimate the area of a square more accurately than a skinny rectangle, and one of the goals of treemaps is to use the area of each rectangle to present some useful attribute.

I define order here to mean that a treemap algorithm is ordered if the rectangles it generates are laid out in a spatial sequence that corresponds to the input sequence. Not all treemap algorithms are ordered, and order is important since it is easier for users to find specific items in ordered displays. Rodden has showed the importance of order in image browsing [19]. In addition, ordered displays make it easier to track items if they change over time since in an ordered display, each item will stay in approximately the same place on the screen.

Until recently, there were no algorithms that provided both properties.

Treemap Related Work

The original treemap algorithm by Shneiderman [21] uses a simple "slice and dice" approach. It divides the input rectangle into a single horizontal or vertical list of rectangles – each one typically being quite skinny. If the algorithm is applied recursively, the sub-rectangle would be split in the opposite orientation as the parent. This algorithm generates ordered rectangles, but they typically have extreme aspect ratios.

An important ensuing treemap algorithm, called squarified treemaps, gave up on ordering, but created rectangles with smaller aspect ratios [10]. Squarified treemaps work by recursively dividing the space in two, and laying out some of the rectangle in one part, and the rest of the rectangles in the other part, where the list of rectangles is split based on optimizing the resulting aspect ratios. A variation of this algorithm was independently developed for SmartMoney's MarketMap applet [4]. Recently, Shneiderman and Wattenberg introduced ordered treemaps [5] which offer a compromise solution where the resulting rectangles are ordered, and somewhat squarified, but do not have as good aspect ratios as those generated by squarified treemaps. Other approaches to space-filling algorithms have been considered but they typically do not have all the nice properties of treemaps, such as that by Harel and Yashchin [13] which does not assign the size of the rectangles to any independent variable.

Treemaps have been applied to a number of domains, from visualizing hard disk usage [3] to the stock market [4]. However, in every current usage of treemaps to date, they are used to visualize a two-dimensional dataset where typically, one dimension is mapped to the area of the rectangles (as computed by the treemap algorithm), and the other dimension is mapped to the color of the rectangle. Then, a label is placed in the rectangles which are large enough to accommodate them, and the user can interact with the treemap to get more information about the objects depicted by the rectangles.

Surprisingly enough, there are not any published uses of treemaps where other information is placed in the rectangles. PhotoMesa appears to be the first application to put images within the area of each treemap rectangle.

There is a good reason why treemaps have not been used in this manner before. This is because while treemaps guarantee that the area of each generated rectangle is proportional to an input number, they do not make any promise about the aspect ratio of the rectangles. Some treemap algorithms (such as squarified treemaps) do generate rectangles with better aspect ratios, but the rectangles can have any aspect ratio. While this is fine for general purpose visualizations, it is not appropriate for laying out images because images have fixed aspect ratios, and they do not fit well in rectangles with inappropriate aspect ratios.

Let us look at applying existing treemap algorithms to laying out fixed size objects, such as images. For now, let us assume without loss of generality that the images are all square. We will see later that this does not affect layout issues. Given a list of groups of images to lay out, the obvious input to the treemap algorithm is the number of images in each group. The treemap algorithm will generate a list of rectangles, that each need the corresponding images to be laid out within.

For each rectangle and group of images, the first step is to decide on the dimensions of a grid with which to lay out the images in the rectangle. Given the aspect ratio of the rectangle, we compute the number of rows and columns that best fit the images.

The resulting grid may have more cells than there are images, but will not have any empty rows or columns. This layout, however, is not guaranteed to fit in the rectangle. For example, consider a rectangle that was computed to hold a single image. It will have an area of 1.0, but could be long and skinny, perhaps with a width of 10.0 and a height of 0.1. The obvious solution is to scale down the images just enough to fit in the bounds of the rectangle.

Herein lies the problem. Since each group of images has to fit in to a separate rectangle, each group of images will have to potentially be scaled down. This will result in each group of images being a different size. Furthermore, since

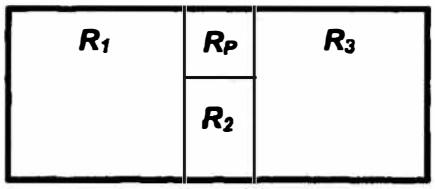

Figure 3: Basic layout strategy of the ordered treemap algorithm. The pivot is layed out in R_P, and $L_1 \ldots L_{P-1}$ are layed out in R_1 while $L_{P+1} \ldots L_n$ are layed out in R_2 and R_3.

the rectangles are arbitrarily sized and positioned, and the images are scaled, the resulting groups of images will not align with each other in a visually attractive way.

It is standard graphic design practice to align content in a way that makes it easy for the eye to quickly scan different areas. If each group of images is a different size and they are not aligned, this will make the resulting layout less attractive, and may make it slower for a user to quickly scan. See Figure 4 for the result of laying out a simple sequence of images using the ordered treemap and quantum treemap algorithms to see the difference in overall layout. Note how with the ordered treemap, group #4 consisting of a single image is scaled much smaller than the other images. With the quantum treemap algorithm, all images are the same size, and all images are aligned on a single grid across all the groups.

Ordered Treemaps

To understand the quantum treemap algorithm, it is necessary to first understand the basics of the ordered treemap algorithm because the former is a direct modification of the latter.

The ordered treemap algorithm, as with all treemap algorithms, take as input and produces output:

Input	$L_1 \ldots L_n$	An ordered sequence of numbers.
	Box	A box to lay out the rectangles within.
Output	$R_1 \ldots R_n$	An ordered sequence of rectangles that completely fill *Box*, and where the area of R_i is proportional to L_i.

The algorithm is similar to QuickSort. It chooses a pivot, L_P, and places it in *Box*. It then recursively lays out $L_1 \ldots L_{P-1}$ on one side of the pivot, and $L_{P+1} \ldots L_n$ on the other side of the pivot. Figure 3 shows the basic visual strategy for a horizontal layout. A corresponding approach is used for a vertical layout.

The ordered treemap algorithm is described in detail in [5], and is summarized here.

1. If $n = 1$, then return a rectangle $R = Box$ and stop.
2. Choose a pivot element, L_P. Pivot selection strategies include picking the middle element or the largest one.
3. Calculate R_1 so that its height fills *Box*, and so that its width is large enough to contain $L_A = L_1 \ldots L_{P-1}$.
4. Split $L_{P+1} \ldots L_n$ into two sublists, L_B and L_C that will be laid out in R_2 and R_3. Calculate where the splitting point is so that R_P has an aspect ratio closest to 1.
5. Calculate R_P, R_2 and R_3. This is performed by using the ratio between the size of the corresponding lists, and breaking up the available space by the same ratios.
6. Recursively apply the ordered treemap algorithm to L_A in R_1, L_B in R_2, and L_C in R_3.

This algorithm results in rectangles that are fairly square, and are ordered approximately left to right (or top to bottom in a vertically oriented box).

Quantum Treemap Algorithm

The goal of the quantum treemap algorithm is similar to other treemap algorithms, but instead of generating rectangles of arbitrary aspect ratios, it generates rectangles with widths and heights that are integer multiples of a given elemental size. In this manner, it always generates rectangles in which a grid of elements of the same size can be laid out. Furthermore, all the grids of elements will align perfectly with rows and columns of elements running across the entire series of rectangles. It is this basic element size that can not be made any smaller that led to the name of *quantum treemaps*.

The quantum treemap (QT) algorithm is based directly on the ordered treemap (OT) algorithm. However, the basic approach could be applied to any other treemap algorithm. QT's input and output are similar to those of OT, but instead of returning a set of rectangles that precisely fill the specified input *Box*, it generates a set of rectangles that only approximate the input *Box*. Because there is some wasted space, the resulting set of rectangles are usually larger than *Box*, but have close to the same aspect ratio. In addition, QT takes an additional input parameter which is the aspect ratio of the elements to be laid out in *Box*.

QT starts in exactly the same manner as OT, picking a pivot, subdividing the space, and recursively applying the algorithm to each sub-space. It works in the same way until step 1 stops the recursion.

At this point (step 1), rather then just unwinding the recursive stack, it adjusts the computed rectangle by modifying its dimensions, making it big enough for precisely the specified number of elements.

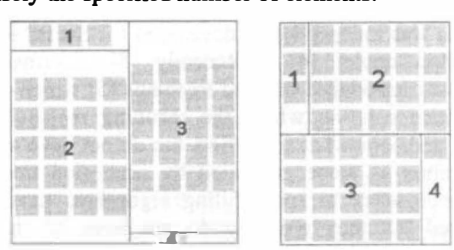

Figure 4: The result of laying out a sequence of 4 groups of elements (of size 3, 20, 20, 1) using ordered treemap (left) and quantum treemap (right)

Then, as the recursion unwinds, the caller must accommodate the generated rectangles which may not fit precisely into the box that was asked for. This is the tricky part, and is captured in a modified version of step 6. Since the rectangles generated by the recursive call may be bigger or smaller in either dimension than was asked for, the rectangles from the other regions must be moved so they don't overlap, and possibly grown so they align nicely with neighboring rectangles. As an example, see Figure 4 (right). Rectangle #4 was originally computed to have dimensions (1x1), but since Rectangle #3 was much taller, Rectangle #4 was stretched to be 4 units tall to match the height of Rectangle #3. Similarly, Rectangle #1 was stretched to match the height of Rectangle #2. The new algorithmic steps are stated here:

new 1. If $n = 1$, then compute a rectangle R that contains exactly L quantums in a grid arrangement that has an aspect ratio as close as possible to that of *Box* and stop.

new 6. Recursively apply the ordered treemap algorithm to L_A in R_1, L_B in R_2, and L_C in R_3.

new 6a. Translate the rectangles in R_P, R_2, and R_3 to avoid overlapping R_1 or each other.

new 6b. Even out the rectangles in the sub-regions in the following manner. Make sure that R_P and R_2 have the same width. Make sure that R_P and R_2 together have the same height as R_1. Make sure that R_3 has the same height as R_1. Each of these evening steps can be accomplished similarly by finding if one of the regions is too small. Then if it is not wide enough, add the extra amount to the width of the rectangles in that region that touch the right boundary of the region. Do the analogous action to rectangles not tall enough.

Element Aspect Ratio Issues

QT assumes that all elements that will be laid out in the rectangles produced by QT are the same aspect ratio, and that aspect ratio is an input parameter to QT. It turns out, however, that it is not necessary to modify the internal structure of QT to accommodate the element's aspect ratio. Instead, the dimensions of the starting box can simply be stretched by the inverse of the element aspect ratio.

Growing Horizontally or Vertically

In step 1, the requested rectangle may be grown to accommodate the quantum element size. There is a basic question of whether to grow this rectangle horizontally or vertically. The simple answer is just to grow in the direction that results in a rectangle that most closely matches the aspect ratio of the original rectangle. However, the algorithm as a whole produces better layouts if it always grows horizontally (or vertically for layout boxes that are oriented vertically).

The issue here is somewhat subtle, but is related to step 6b where the rectangles are evened. If, for example, rectangles in R_3 are made taller, than all of R_1 and R_2 will have to made taller as well to match R_3. If instead, the rectangles in R_3 are made wider, than only the other rectangles in R_3 will need to be made wider, and the rectangles in R_1 and R_2 can be left alone.

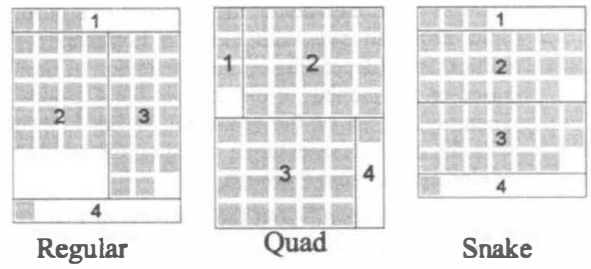

Figure 5: The result of applying the three stopping conditions to a sequence of 4 groups of elements (of size 3, 20, 20, 1).

In general, the evening aspect of the QT algorithm remains somewhat problematic. While it works well for most data sets, it occasionally yields undesirable layouts due to too much wasted space. This can happen when one region ends up growing a fair amount to accommodate data that doesn't happen to fit the starting rectangles, and then the other regions have to be grown to match. When these other regions are grown to match, the resulting rectangles are bigger than necessary, and there is wasted space. This doesn't seem to be a problem for datasets unless they contain many regions with a very small number of elements (< 10). In practice, it has not been a significant problem for the real image datasets I have viewed, although sometimes there is a little more wasted space than I would like.

Stopping Condition Improvements

Changing the stopping conditions and offering special layouts for a small number of special cases can produce substantially better total results. The new stopping conditions apply equally to QT as well as to OT.

The improvement is because the layout of rectangles depicted in Figure 5 (left) is not necessarily the one with the smallest aspect ratios. In addition, it generates a layout that is somewhat difficult to parse visually because the eye has to move in 3 directions to focus on the 4 rectangles (vertically from #1 to #2, horizontally from #2 to #3, and then vertically from #3 to #4).

The layout can be improved, and visual readability by offering two alternative layouts. The first produces a "*quad*" of (2x2) rectangles. The second produces a "*snake*" layout with all 4 rectangles laid out sequentially – either horizontally or vertically. The snake layout can be equally well applied to 2, 3, or more rectangles. PhotoMesa applied it up to 5 rectangles. Figure 5 shows the result of laying out a sequence of 4 groups of elements using the three strategies. The new algorithmic step is:

new 1a. If $n = 4$, then first try the regular layout by continuing and letting the recursion get down to the bottom level

new 1b. If $n = 4$, then layout the 4 groups in a quad. Split *Box* into two with either a horizontal or vertical split (depending on the orientation of *Box*) based on the number of elements in the 4 groups. Then, split each of the remaining boxes in two with the opposite orientation based on the number of elements in those 2 groups.

new 1c. If n = 4, then layout the 4 groups in a snake by dividing *Box* into 4 sub boxes (horizontally or vertically, depending on the orientation of *Box*), based on the number of elements in the 4 groups.

new 1d. Compute the aspect ratios and wasted space of the 4 resulting rectangles from steps 1a, 1b, and 1c, and use the layout with the best overall results.

Since no one layout strategy always gives the best result for all input data, for 5 or fewer rectangles, PhotoMesa computes layouts using all strategies (original, quad, and snake) and picks the best one. In practice, this strategy produces layouts with substantially squarer aspect ratios. Running 100 randomized tests with 100 rectangles, and random numbers of elements per rectangle, ranging from 10 to 1000 produced an average aspect ratio of 3.92 with the original stopping conditions, and 2.68 for the new stopping conditions.

Pivot Selection Improvement

In addition to the two pivot selection strategies discussed in [5], a third strategy specifically targeted at the evening problem previously discussed offers improved results. The new strategy, called "*split size*" gives better results for some input data.

The basic approach is to pick the pivot that will split the lists of elements into equal sizes, or as close to equal as possible. With the sublists containing similar numbers of elements to lay out, there tends to be less evening, and therefore less wasted space. The new algorithmic step is:

new 2. Choose a pivot element, R_P. Pivot selection strategies include picking the middle element, the largest one, or the one that results in splitting the elements into lists that are as close to equal size as possible.

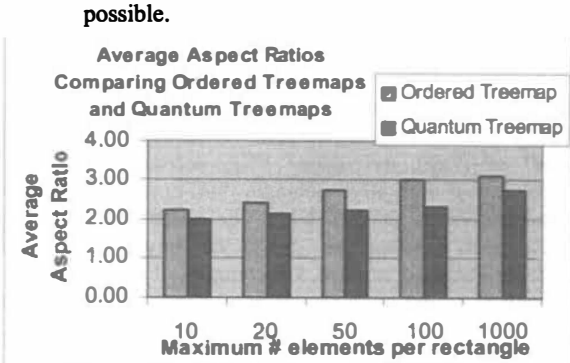

Figure 6: Average aspect ratio of all rectangles run on both algorithms with 100 rectangles with random numbers of elements per rectangle.

No single pivot selection strategy always works best, so in practice, PhotoMesa computes the layouts with all three pivot selection strategies, and picks the best one based on the average aspect ratios of the resulting rectangles, and the amount of wasted space.

Quantum Treemap Analysis

One of the basic characteristics of QT is that it works better when there are more elements per group. This is because it gives the algorithm more flexibility when computing rectangles. A rectangle of, say, 1000 elements, can be arranged in quantified grids of many different sizes such as (30x34), (31x33), (32x32), etc. – each of which use the space quite efficiently. Rectangles containing smaller numbers of elements, however, do not offer as many options, and often use space less efficiently. For example, a rectangle containing 5 elements can be laid out in (1x5), (2x3), (3x2), or (5x1). These four options do not give the algorithm as much flexibility as the dozens of grid options afforded by the larger number of elements.

In order to assess the effectiveness of QT, it was compared to OT with a series of trials using random input. Each algorithm was run 100 times generating 100 rectangles with the number of elements in each rectangle being randomly generated. This was done for 5 different ranges of the number of elements per rectangle. For each test, the average aspect ratio of all the rectangles was recorded as well as the space utilization, which was recorded as the percentage of space not used to display elements (wasted space). The same random numbers were used for each algorithm. Figures 6 and 7 show the results of these tests. Quantum treemaps did better in terms of aspect ratio, and ordered treemaps did better in terms of wasted space.

However, the crucial visual advantage of QT is that it always produces layouts where elements are the same size and are aligned on a single global grid.

BUBBLE MAPS

While quantum treemaps work well, they waste some space. While the use of some white space can be helpful to visually distinguish the groups, too much white space

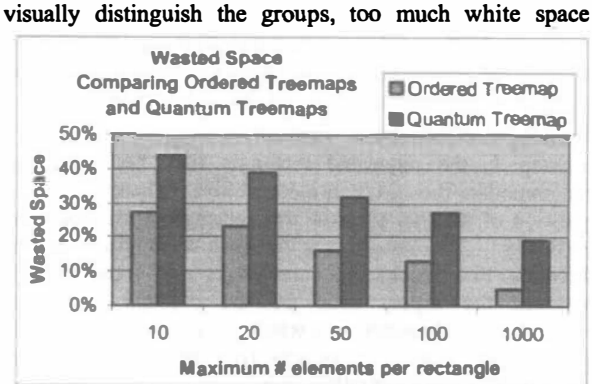

Figure 7: Average wasted space as a percentage of the entire layout space available. Tests run on both algorithms with 100 rectangles with random numbers of elements per rectangle.

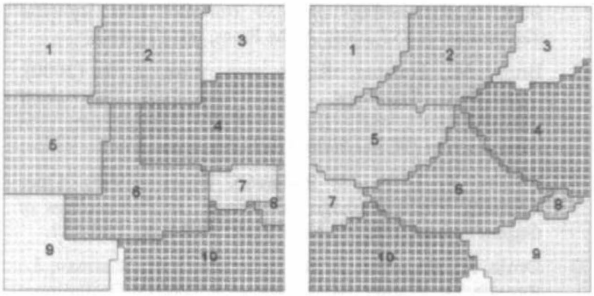

Figure 8: Bubblemap layouts of 10 groups of up to 200 rectangles. Rectangular (left) and circular (right).

becomes unattractive and wasteful. While it may be possible to improve the quantum treemap algorithm, it is impossible to lay out images in a rectangle without sometimes leaving unused space. An alternative approach is to give up on the idea that the space must be divided into rectangles, and instead allow more complex shapes.

Bubblemap is a new algorithm that lays out groups of quantum-sized objects in an ordered layout with no wasted space per group, although there is some wasted space for the entire area. The groups of objects can be created in different shapes, such as rectangular or circular, but the groups of objects only approximate those shapes, rather than define them exactly. Figure 8 shows a rectangular and a circular bubblemap layout of 10 groups of up to 200 rectangles per group. The bubblemap algorithm has also been integrated into PhotoMesa as a user-selectable layout option. Figure 9 shows the bubblemap algorithm applied to a set of images in PhotoMesa. There is no wasted space, but the regions have arbitrary shapes.

A more sophisticated approach to laying out related images in a grid has been pursued by Basalaj with his *Proximity Grid* algorithm [7]. It takes a set of objects with a high-dimensional set of relationships and generates a grid layout of those objects so that similar objects will be near each other on the grid. Bubblemaps, on the other hand, are much simpler and assumes the input is pre-clustered. They keep the clusters of images together, rather than optimizing an n-dimensional set of relationships.

The bubblemap algorithm is completely different than the treemap algorithm. Rather than subdividing rectangles, it is based on a standard pixel-based bucket fill algorithm. It works by filling cells in a grid, keeping track of which cells get assigned to images from which group. It fills the cells one group at a time. By using different algorithms to select the next cell to fill, the shape of the groups can be controlled. The basic algorithm runs in $O(n)$ time for n images. The basic algorithm follows:

Input: $L_1 \ldots L_n$, Aspect Ratio

1. Compute the size of the overall grid based on total number of images to layout, and the desired resulting aspect ratio.
2. Create a grid of size computed from step 1, and set each cell to the value UNASSIGNED.
3. For each group of images, L_i, call the fill algorithm, starting at step 4, and then stop.
4. Find the starting point to fill by looking for the first UNASSIGNED cell in the grid (in left-right, top-bottom order). Initialize a list of cells, called LIST, and add the starting point to LIST.
5. If LIST is empty than stop. Else, take the first element, P, off of LIST, and set the cell at P's location to the value ASSIGNED.
6. For each UNASSIGNED neighbor of P, Q, add Q to LIST, and set the cell at Q's location to the current group ID. GOTO 5. Note that the order in which the neighbors are added to LIST affect the shape of the resulting groups.

Bubblemaps are easy to implement, use space efficiently, and give ordered space-filling layouts like treemaps do. However, since they produce arbitrary-shaped regions rather than rectangles, it is likely that users will find them harder to parse visually. So, while they may be appropriate for applications where the goal is simply to have related items near each other, they may not be as appropriate for tasks where users need to distinguish clearly between groups.

FUTURE WORK

The PhotoMesa application has a long list of features to be added. An important one is to integrate it with PhotoFinder and other sources of metadata. Another area to look into is deploying PhotoMesa in different modalities, such as stand-alone CDs for sending to family and friends, running on a kiosk for museums or lobbies, and integration with the Web. The zoomable characteristics of PhotoMesa make it a good match with the Web, offering a potentially efficient manner to browse large image databases on the Web since only the resolution needed for the current view needs to be sent to the client. And of course, a detailed user study is needed to understand how the zoomable interaction of PhotoMesa compares to more traditional approaches, and which layout techniques are most effective.

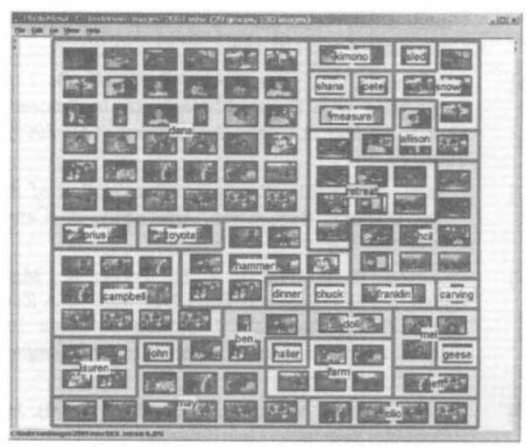

Figure 9: The same images as in Figure 3, but laid out with a rectangular bubblemap.

CONCLUSION

This paper presents PhotoMesa, a zoomable image browser, and two new algorithms for laying out groups of images or other fixed-size visual objects. The primary innovations are: 1) a simplified set of interactions for navigating through a set of objects in a zoomable user interface; and 2) algorithms to lay out fixed-size objects, such as images, in two-dimensional space, automatically creating groups for related objects.

By bringing together the aforementioned innovations with existing zoomable user interface technology, PhotoMesa offers a significant advance in the ability to comfortably browse large numbers of images. Based on its initial popularity and enthusiastic feedback, PhotoMesa appears to have satisfied its initial design goals of being simple to use in a family setting, requiring no setup time, and naturally supporting co-present use.

ACKNOWLEDGEMENTS

I appreciate the feedback on PhotoMesa by many HCIL members. In particular, thanks to Jesse Grosjean who suggested the approach taken in the bubblemap algorithm. In addition, I thank Ben Shneiderman for suggesting the interactive textual list of groups, to Allison Druin for suggesting the ability to color groups, to Jon Meyer and Catherine Plaisant for advise on the visual design of PhotoMesa, to Matthias Mayer for first suggesting I try to display several directories of images at once, and to Mark Stefik from Xerox PARC for suggesting the magnified preview images. Finally, I appreciate Susanne Jul's excellent editorial comments on this paper.

REFERENCES

[1] ACDSee (2001). http://www.acdsystems.com/english/products/acdsee/acdsee-node.htm.

[2] Canon ZoomBrowser (2001). http://www.powershot.com/powershot2/software/ps_pc_view.html.

[3] DiskMapper (2001). http://www.miclog.com/dmdesc.htm.

[4] SmartMoney MarketMap (2001). http://www.smartmoney.com/marketmap/.

[5] Baker, M. J., & Eick, S. G. (1995). Space-Filling Software Visualization. *Journal of Visual Languages and Computing, 6*, pp. 119-133.

[6] Balabanovic, M., Chu, L. L., & Wolff, G. J. (2000). Storytelling With Digital Photographs. *In Proceedings of Human Factors in Computing Systems (CHI 2000)* ACM Press, pp. 564-571.

[7] Basalaj, W. (2000). *Proximity Visualization of Abstract Data*. Doctoral dissertation, University of Cambridge, Cambridge, England.

[8] Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., & Furnas, G. W. (1996). Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing, 7*, pp. 3-31.

[9] Bederson, B. B., Meyer, J., & Good, L. (2000). Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. *In Proceedings of User Interface and Software Technology (UIST 2000)* ACM Press, pp. 171-180.

[10] Bruls, M., Huizing, K., & van Wijk, J. J. (2000). Squarified Treemaps. *In Proceedings of Joint Eurographics and IEEE TCVG Symposium on Visualization (TCVG 2000)* IEEE Press, pp. 33-42.

[11] Combs, T. T. A., & Bederson, B. B. (1999). Does Zooming Improve Image Browsing? *In Proceedings of Digital Library (DL 99)* New York: ACM, pp. 130-137.

[12] Druin, A., Bederson, B. B., Hourcade, J. P., Sherman, L., Revelle, G., Platner, M., & Weng, S. (2001). Designing a Digital Library for Young Children: An Intergenerational Partnership. *In Proceedings of Joint Conference on Digital Libraries (JCDL 2001)* ACM Press, pp. pp. 398-405.

[13] Harel, D., & Yashchin, G. (2000). An Algorithm for Blob Hierarchy Layout. *In Proceedings of Advanced Visual Interfaces (AVI 2000)* ACM Press, pp. 29-40.

[14] Hascoët-Zizi, M., & Pediotakis, N. (1996). Visual Relevance Analysis. *In Proceedings of International Conference on Digital Libraries (DL 96)* ACM Press, pp. 54-62.

[15] Jul, S., & Furnas, G. W. (1998). Critical Zones in Desert Fog: Aids to Multiscale Navigation. *In Proceedings of User Interface and Software Technology (UIST 98)* ACM Press, pp. 97-106.

[16] Kang, H., & Shneiderman, B. (2000). Visualization Methods for Personal Photo Collections Browsing and Searching in the PhotoFinder. *In Proceedings of IEEE International Conference on Multimedia and Expo (ICME2000)* New York: IEEE, pp. 1539-1542.

[17] Platt, J. (2000). AutoAlbum: Clustering Digital Photographs Using Probabalistic Model Merging. *In Proceedings of IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL-2000)* IEEE Press,

[18] Robertson, G. G., & Mackinlay, J. D. (1993). The Document Lens. *In Proceedings of User Interface and Software Technology (UIST 93)* ACM Press, pp. 101-108.

[19] Rodden, K., Basalaj, W., Sinclair, D., & Wood, K. (2001). Does Organisation by Similarity Assist Image Browsing. *In Proceedings of Human Factors in Computing Systems (CHI 2001)* ACM Press, pp. 190-197.

[20] Shen, C., Moghaddam, B., Lesh, N., & Beardsley, P. (2001). Personal Digital Historian: User Interface Design. *In Proceedings of Extended Abstracts of Human Factors in Computing Systems (CHI 2001)* ACM Press,

[21] Shneiderman, B. (1992). Tree Visualization With Treemaps: A 2-D Space-Filling Approach. *ACM Transactions on Graphics, 11*(1), pp. 92-99.

[22] Shneiderman, B., & Kang, H. (2000). Direct Annotation: A Drag-and-Drop Strategy for Labeling Photos. *In Proceedings of IEEE Conference on Information Visualization (IV2000)* New York: IEEE, pp. 88-98.

[23] Stewart, J., Bederson, B. B., & Druin, A. (1999). Single Display Groupware: A Model for Co-Present Collaboration. *In Proceedings of Human Factors in Computing Systems (CHI 99)* ACM Press, pp. 286-293.