Hierarchical Layouts for Photo Libraries

Jack Kustanowitz and Ben Shneiderman
University of Maryland

We use an annotated digital photo collection to demonstrate a two-level auto-layout technique consisting of a central primary region with secondary regions surrounding it. Because the object sizes within regions can only be changed in discrete units, we refer to them as quantum content. Our real-time algorithms enable a compelling interactive display as users resize the canvas, or move and resize the primary region.

Automatic text-flow algorithms have brought about a revolution in the page-layout process. Writers and publishers now take for granted that text will break correctly at the end of a line, or flow around a drawing within a document.[1] Today's Web browsers work on this same principle of dynamic flow, and most Web pages will reflow the text as the window is resized. If users print a document, properly configured text will reflow to meet the paper's dimensions.

This article explores the possibility of performing rapid, automatic layout with nonoverlapping, 2D fixed-aspect-ratio objects, such as photos. These objects can appear in a central primary region, which defines the entire layout, or in secondary regions that offer lower-level detail, such as parents in the primary region, surrounded by secondary regions containing many pictures of each of their children. We address such common two-level categories in this article, leaving more complex layouts for future work. In all cases, the primary region highlights one photo, concept, or text block; the secondary regions serve the purpose of showing organized subcollections that thematically relate to the primary region.

The techniques presented here apply especially to photo libraries, but might have variants that apply to newspaper layouts, online magazines, and electronic product catalogs. A compelling aspect of these algorithms is the interactive redisplay as users change the rectangular canvas size and shape, add or delete secondary regions, and add or delete objects to a secondary region. We combine these algorithms and refer to them holistically as the *bilevel radial quantum*, or *BRQ* algorithm—pronounced as "brick," not only for pronunciation of the acronym, but also because of its playful resemblance to a brick wall. This engaging animated interaction (see the "Dynamic behavior" section) is an important feature for consumer applications such as personal photo management.

Case study: Photo layout

In many commercial photo management tools—such as ACD's ACDSee, Adobe's Photoshop, Apple's iPhoto, and Google's Picasa—photos are presented in a simple grid. Because all of the photos are the same size we refer to them as *quantum content*. While these programs allow advanced metadata annotation, such annotation doesn't carry over to the presentation mode so that users could view photos with different annotations simultaneously, sorted appropriately.

Web-based commercial stock photo vendors—such as Punchstock, Corbis, and Getty—also use simple grid layouts to show search results, even though they often have rich metadata that would support more orderly presentations.

Our work extends beyond the design of tools such as PhotoMesa (designed by the University of Maryland's Human–Computer Interaction Laboratory), which show the benefits of a grid layout based on the directory structure or metadata annotations. In this article, we present several screen shots with accompanying descriptions that illustrate how such a bilevel layout presents interesting possibilities for this domain. More extensive justification and a user study appear elsewhere.[2]

We generated all of our screen shots with the BRQLayer tool, which implements the algorithms we describe. The program and source code are freely available at http://www.cs.umd.edu/hcil/brqlayer. We created each layout by choosing several tags from the annotated collection (for example, family members Al, Shuly, Esther, Simmy, and Lani in Figure 1), and then choosing a representative photo for the primary region in the center. The layouts were then sized to the desired dimensions, and the primary region was filled and scaled to the desired location and size. Throughout, the thumbnails in the secondary regions maintained a maximal size as they resized and moved themselves in response to changes in the primary region; no user input was required in creating the regions or choosing the thumbnail size.

Family photo collection

Figure 1 depicts a family layout, in which the user has chosen photos for each family member.

1070-986X/06/$20.00 © 2006 IEEE

Published by the IEEE Computer Society

In it, similar quantities of photos are in each region, which lends itself to a balanced view. Layouts with reasonably similar quantities should be able to minimize wasted space, and with some manipulation of the size and position of the primary region, the user can create a layout that's efficient and attractive.

Vacation trips

Figure 2 depicts a generated layout of photos taken during a trip to Italy. This is a case in which the ordering is important, so that users can readily find a part of the trip that was toward the beginning, middle, or end. This is also a good example of having more than four photo regions, along with the need for dynamic and proper balancing. While other timeline-generation utilities exist (Picasa has a particularly appealing interactive timeline), this application allows the dynamic creation of a printable layout that shows the entire photo collection, supplemented with timeline information.

Organization charts

A common way of grouping is to represent organizational hierarchies, but they don't usually have a sequential layout requirement. Figure 3 (next page) shows six regions representing some research areas of the University of Maryland's Computer Science Department with a photo for faculty members. The primary region shows the department chair. In this case, we chose a layout that contained enough horizontal blank space for the titles. As an example, if the primary region were enlarged to the left, the thumbnail size could remain constant, but the text "Scientific Computing" would get truncated.

Real estate browser

Figure 4 presents a hypothetical real estate Web site, containing a bilevel radial quantum photo layout within a Web page. As users change the browser window size, the layout, regions, and photos all size dynamically to allow for the largest photo size possible given the relative number of photos in each area. In this way, the photo flow resembles text reflowing or resizing.

Should a new house come on the market in Silver Spring, it would fit nicely in the blank space under the houses currently visible. A new house in one of the other regions could push the photo size smaller, or might instead cause the secondary regions to be distributed differently about the primary region, possibly without requiring smaller photos.

Problem definition and requirements

As we defined the problem, certain requirements became clear. For example, we wanted the canvas size to remain stable, even if an algorithm could discover an improved layout by reducing or enlarging the canvas dimension. Similarly, uniform size for all photo thumbnails was an important goal, even if an algorithm could

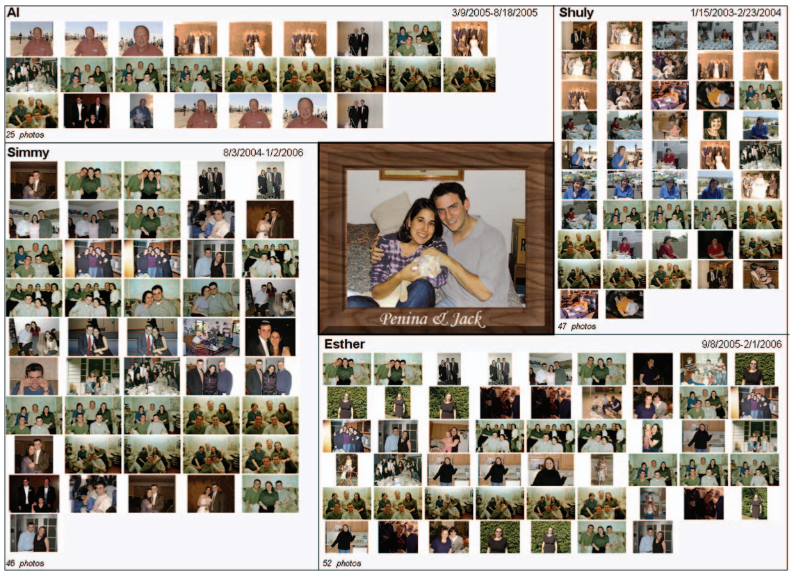


Figure 1. Family photo collection depicting the main subjects in a primary region surrounded by grouped photos of family members. Even a collection of this size can be resized and manipulated in real time.

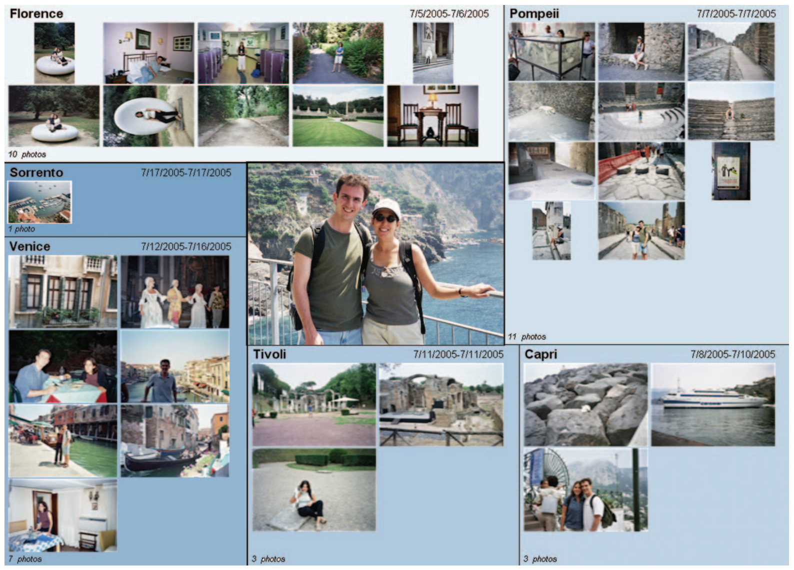

Florence 7/5/2005-7/6/2005
10 photos
Pompeii 7/7/2005-7/7/2005
11 photos
Sorrento 7/17/2005-7/17/2005
1 photo
Venice 7/12/2005-7/16/2005
7 photos
Tivoli 7/11/2005-7/11/2005
3 photos
Capri 7/8/2005-7/10/2005
3 photos

Figure 2. Honeymoon in Italy showing six locations during a two-week trip. The radial increase in shading along with the date captions reinforce the sequential nature of this series.

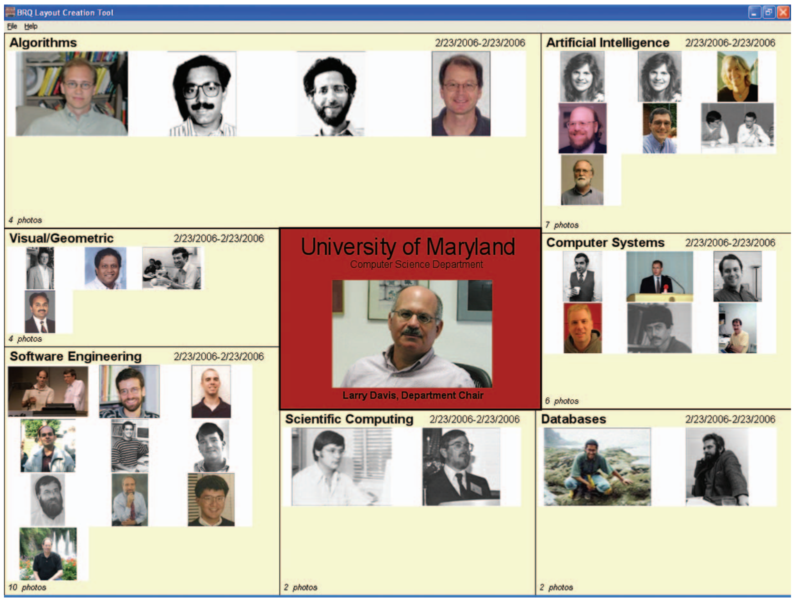
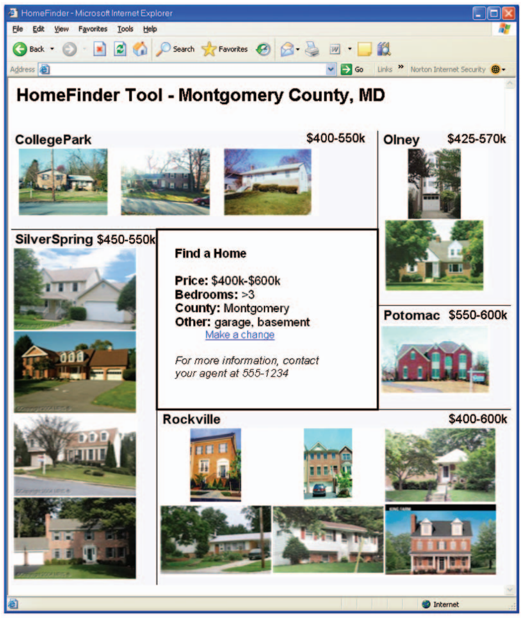

BRQ Layout Creation Tool
File Help
Algorithms 2/23/2006-2/23/2006
4 photos
Artificial Intelligence 2/23/2006-2/23/2006
7 photos
Visual/Geometric 2/23/2006-2/23/2006
4 photos
University of Maryland
Computer Science Department
Larry Davis, Department Chair
Computer Systems 2/23/2006-2/23/2006
6 photos
Software Engineering 2/23/2006-2/23/2006
10 photos
Scientific Computing 2/23/2006-2/23/2006
2 photos
Databases 2/23/2006-2/23/2006
2 photos

Figure 3. University of Maryland's Computer Science Department organization chart with seven research areas. In this layout, the constant thumbnail size was enforced, causing a layout that doesn't give artificial prominence to any group of photos, at the expense of unused space in each region.


HomeFinder - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address Go Links Norton Internet Security
HomeFinder Tool - Montgomery County, MD
CollegePark $400-550k
Olney $425-570k
SilverSpring $450-550k
Potomac $550-600k
Rockville $400-600k
Internet

Find a Home

Price: $400k-$600k
Bedrooms: >3
County: Montgomery
Other: garage, basement
Make a change

For more information, contact your agent at 555-1234

Figure 4. Real estate browser with five communities and the price ranges of selected homes. This illustrates a potential use of the BRQ algorithm on the Web, which depends heavily on real-time page rendering and dynamic reflowing as the browser is resized.

reduce unused space by reducing or enlarging some thumbnails. Future research might relax some of the requirements or produce interesting variations that are potentially applicable in other domains. Earlier work on layout requirements for photos and interfaces provided guidance for our work[1,3-5] (see also the "Related Work" sidebar).

Fixed canvas size

The algorithm can't change the canvas size (width by height, usually the window size chosen by users). As users vary the canvas size (the algorithm runs again with each resize), the layout will change, still filling the canvas exactly, never causing it to grow or shrink under algorithmic control.

Relaxing this requirement would let the canvas size grow if the algorithm determined that a better layout could be achieved if the canvas were 5 percent wider, for example. While possibly useful, this introduces a feedback loop in which the canvas size determines layout, which in turn determines canvas size. Preserving this requirement allows the algorithm to be deterministic, avoids nonlinearity, and maintains the user's sense of control.

Fixed primary region size and location

The user sets the primary region's size and location; the algorithm doesn't modify this. If users change the primary region's size and location, then the layout will be updated to reflect the new size and position. As we mentioned in the "Fixed canvas size" section, relaxing this requirement also introduces a feedback loop that we avoid for similar reasons.

Uniform quantum size and aspect ratio

Initially, all of the quanta (in the test case, photo thumbnails) must be the same size and aspect ratio. Photos that are intrinsically different sizes can be placed on a background that's constant, so that some photos take up the whole background and on some the background shows through on the margins. Once the layout is in place, this requirement can be removed to minimize wasted space in each region. This should occur as a later step to avoid feedback in the initial layout algorithm.

Secondary regions distributed in quadrants

We based the algorithms on the idea of having four fixed quadrants surrounding the primary region (see Figure 5a on p. 66). Within each quadrant we can place several secondary regions, but no secondary region can cross a quadrant boundary. Additionally, each quadrant must contain at least one region. The benefit is a tight alignment of secondary regions with the edges of the primary region, thus making a visually appealing layout that enables easy-to-scan grids of photos in each secondary region.

Related Work

Our work extends the work on ordered and quantum treemaps[1] applied in PhotoMesa. Treemaps map a hierarchy onto a rectangular region in a space-filling manner; ordered treemaps add an additional requirement of maintaining order in the layout.

Order-preserving layouts reduce the effect of rectangles shifting position as the item sizes change. The quantum requirement is useful in photo layouts, in which each rectangle has subrectangles of a fixed size. The layout algorithms described in this article add a central (both in importance and in position) rectangle, which can itself be sized and moved around the layout, with the other rectangles in turn rapidly moving and resizing. This bilevel property gives prominence to a certain layout feature, whether it's a graphic, descriptive text, or central feature of a nonphoto application.

While our approach is to define the interaction of requirements in a way that guarantees a solution, other creative ways to solve problems with interrelated constraints include using Monte Carlo methods or genetic algorithms.[2,3] Genetic algorithms have been used to solve image placement problems in the context of automated page layout in a digital album, using principles such as balance, spacing, emphasis, and unity.[4] These algorithms produce useful results that are appropriate in cases where the constraints in question are fuzzy or subjective. In our work they're more quantifiable, which allows for simpler and faster algorithms.

In this article, we present a radial algorithm for two-level layouts. Multilevel radial animations are described elsewhere,[5] in the context of navigating an expanding and contracting graph that works based on polar coordinates. It's possible that this work could be expanded either to higher-level layouts, or that a collection of photos could be navigated using a radial animation to get from one two-level layout to a related one.

New presentation media could drive the demand for alternate 2D dynamic layout techniques. The Personal Digital Historian (PDH),[6] for example, describes a tabletop display several times larger than the screens currently used to view photo collections. While PDH groups similar photos based on user modeling, the algorithms described here could provide an alternate way to dynamically adjust the layout of a tabletop or other large displays based on the user's position at the table or category of interest. On still larger wall-sized displays, thousands of photos could be accommodated, making the utility of meaningful organization still greater. Even in a smaller, nontraditional display,[7] our work could allow a control panel to be smoothly moved around the screen while the surrounding photos resized and flowed accordingly.

Our layout algorithm's final result is that it provides a "focus + content" display, similar to the semantic fisheye view (SFEV) described elsewhere.[8] The SFEV also uses image annotation data to generate a layout based on relationships between terms and similarities in the annotation text. Similar to the SFEV layouts, we can engineer the layouts produced in our work to have a fisheye look, with the advantages of focusing the viewer's attention on a primary feature while still showing related images as being connected, if at a smaller size.

The Bramble system[9] describes a toolkit for a constraint-based layout, using a differential, continuous approach. This allows for dynamic enabling or disabling of specific constraints, potentially in a 3D space. While we describe in the "Problem definition and requirements" section on the problem requirements that resemble constraint-based programming to a degree, our focus was on real-time, smooth, 2D motion to provide maximum interactivity. Generic constraint-based problem solvers require large memory capacity and slow down as the problem size increases, while the domain-specific algorithms we chose often provide real-time performance at the expense of generality.[10] By avoiding backtracking in the algorithms and using today's fast processors, we succeeded by simply recomputing the layout when the parameters changed.

References

1. B. Bederson, B. Shneiderman, and M. Wattenberg, "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies," *ACM Trans. Graphics*, vol. 21, no. 4, 2002, pp. 833-854.
2. W. Graf, "Constraint-Based Layout Framework LayLab and Its Applications," *Proc. ACM Workshop on Effective Abstractions in Multimedia*, ACM Press, 1995; http://www.cs.uic.edu/~ifc/mmwsproc/graf/mm95.html.
3. L. Purvis et al., "Creating Personalized Documents: An Optimization Approach," *Proc. ACM Symp. Document Engineering*, ACM Press, 2003, pp. 68-77.
4. J. Geigel and A. Loui, "Using Genetic Algorithms for Album Page Layouts," *IEEE MultiMedia,* vol. 10, no. 4, 2003, pp. 16-27.
5. K. Yee et al., "Animated Exploration of Graphs with Radial Layout," *Proc. IEEE Symp. Information Visualization*, IEEE CS Press, 2001, pp. 43-50.
6. B. Moghaddam et al., "Visualization & Layout for Personal Photo Libraries," *Int'l Workshop Content-Based Multimedia Indexing*, 2001; http://www.merl.com/publications/TR2001-028.
7. M. Balabanovic, L. Chu, and G. Wolff, "Storytelling with Digital Photographs," *Proc. Conf. Human Factors in Computing Systems,* ACM Press, 2000, pp. 564-571.
8. P. Janacek and P. Pu, "An Evaluation of Semantic Fisheye Views for Opportunistic Search in an Annotated Image Collection," *Int'l J. Digital Libraries*, vol. 5, no. 1, 2005, pp. 42-56.
9. M. Gleicher, "A Graphics Toolkit Based on Differential Constraints," *Proc. 1993 ACM Symp. User Interface Technology*, ACM Press, 1993, pp. 109-120.
10. W. Hower and W. Graf, "Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey," *Proc. Int'l Workshop on Constraints for Graphics and Visualization* (CGV 95), 1995; http://citeseer.ist.psu.edu/hower95research.html.

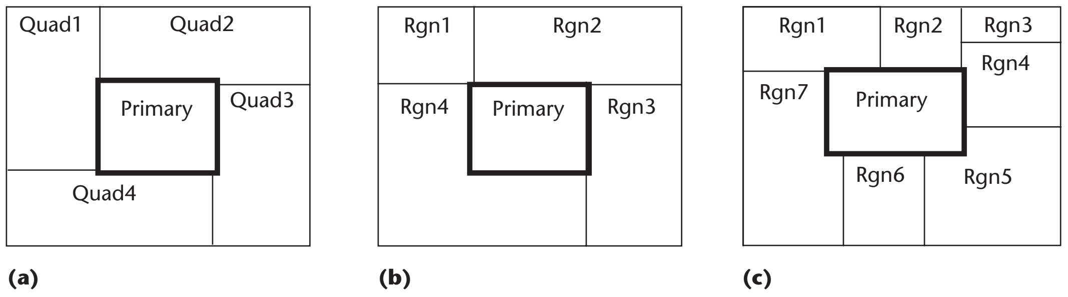


Figure 5. (a) Four quadrants distributed about a primary region. This is our chosen solution. (b) Awkward L-shape caused by using octants for secondary regions. Our solution prevents this. (c) Misalignments caused by arbitrary region placements. Our solution prevents this.

The alternatives are either to tighten this requirement or to remove it entirely. Tightening to eight octants (eight boxes surrounding the primary region) would maintain alignment with the primary region to some degree, but would allow degenerate cases with L-shaped and C-shaped regions, which could pose scanning problems as the viewer tries to determine if the photos should be viewed left-to-right, up-and-down, and so on (see region 4 in Figure 5b). In the case of quadrants, this is only a problem if there are fewer than four regions, and it can be avoided by forcing the primary region to a corner in these cases. For octants, L- and C-shaped regions remain a possibility for up to eight regions, which covers many of the most common scenarios.

Removing the quadrant requirement entirely complicates things even further, as C-shapes and L-shapes would be allowed with no guarantees of lining edges up with the primary region (see Figure 5c). Trying to place regular thumbnails in regions 5 or 7 of Figure 5c could prove difficult. Additionally, layouts causing irregularly shaped regions make it difficult to provide an easily scannable photo grid.

Fixed number of thumbnails

The number of thumbnails in a given region is fixed, in that the algorithm cannot arbitrarily add or remove thumbnails to create a more balanced layout. If one region has dramatically more thumbnails than another, an application might find it advantageous to allow a region to have only *x* times the number of thumbnails of another, or to specify a minimum thumbnail size which (given a fixed canvas size) necessarily changes the number of thumbnails that are visible. In these cases, a scrollbar or a "More" button could be added to view the thumbnails that were left out.

For the purpose of this discussion, we let users decide on the number of thumbnails, but once that decision is made, the number of thumbnails that the algorithm works with is fixed. Allowing the algorithm to internally fine-tune these parameters introduces the undesirable feedback loop discussed previously.

Fixed order of regions

The regions should be laid out in the order in which they are added. This is useful in applications where order is important, whether it's alphabetic, the age of children, or a numeric page order for a table of contents. Preserving order offers the additional benefit of stability when resizing, because it's distracting and undesirable to have regions jumping around when the canvas is resized.[6]

Bilevel radial quantum layout

Based on the requirements that we discuss in the "Problem definition and requirements" section, the following algorithms (that are part of BRQ) will generate regions with the largest possible thumbnail size. We describe the layouts as bilevel to indicate our solution is for two-level hierarchies, as radial to indicate that the secondary regions wrap around the primary region in an ordered manner, and as quantum to indicate that the items in each region are fixed in size and shape.

BRQ layout algorithm

The BRQ layout algorithm comprises three steps:

1. Distribute the secondary tiles among the four quadrants, using the RegionSplit algorithm (see Figure 6).

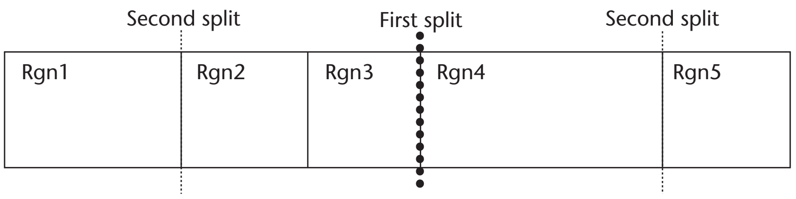


Figure 6. Distribution of five secondary regions among the four quadrants.

2. Set the initial quantum width and height, which is guaranteed to be an upper bound on the possible quantum dimensions, using the InitialQuantumDim algorithm.

3. Reduce the quantum dimensions (keeping the same aspect ratio) until there is no overflow, using the ReviseQuantumDim algorithm. If the quantum dimensions drop below a specified minimum, handle the layout as a special case.

RegionSplit algorithm. This algorithm optimally divides regions up among the four quadrants and works as follows:

1. Choose the region n ($0 < n =$ NumRegions) such that the sum of the thumbnails in all regions up to and including region n is closest to 1/2 of the total number of thumbnails in all regions. This will split the regions into two groups.

2. Do Step 1 for each of the two groups, generating a total of four groups.

The RegionSplit algorithm requires three calls, the first splitting the entire collection of regions in two, and then once again on each half resulting from the first step. Thus, the time for RegionSplit scales linearly with the number of secondary regions.

InitialQuantumDim algorithm. This algorithm sets initial dimensions for maximal thumbnail size. This means that the algorithm's output (*tlength*, *twidth*) is defined to be an upper bound on the thumbnails' size. In the (rare) case where every region has exactly *rows* × *columns* thumbnails, this will also be the final thumbnail size. In every other case, the thumbnail size will need to be reduced until there is no overflow. The algorithm works as follows:

1. Create four rectangles with the same dimensions as the quadrants, as in Figure 7.

2. Let $|t_i|$ $\{1 < i = 4\}$ = the total number of thumbnails in the quadrant by summing the total number of thumbnails in each region in the quadrant.

3. If the thumbnails must exactly fit the space, then the following would be true for each quadrant:

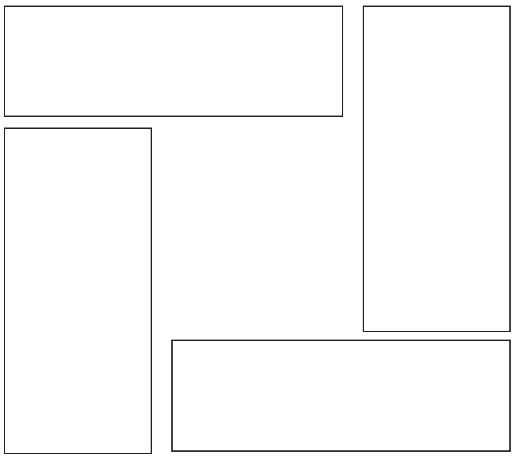


Figure 7. Quadrant rectangles for computing an upper bound of the thumbnail size.

$theightCandidate * twidthCandidate * |t_i| = clientArea_i$

4. Solve for quadrant 1, using the known aspect ratio of the thumbnails:

$theightCandidate = Sqrt(clientArea_1 * thumbnailAspectRatio/|t_1|)$

$twidthCandidate = theightCandidate/thumbnailAspectRatio$

These two values are now upper bounds on the thumbnail size (the "Optimality discussion" section explains why).

5. Solve for the other three quadrants, reducing *theightCandidate* and *twidthCandidate* if lower values are found.

The InitialQuantumDim algorithm is simply a set of calculations for each of the four regions, and thus operates in constant time.

ReviseQuantumDim algorithm. This algorithm takes as its input dimensions that are an upper bound on the possible thumbnail dimensions. It revises those dimensions downward until there are no more columns (quadrants 2 and 4) or rows (quadrants 1 and 3) than any of the quadrants has room for. The algorithm operates as follows:

1. For quadrant 1, determine the number of columns used by the first region as follows:

$columns = \text{ceiling}(quadrantClientWidth/tWidthCandidate)$

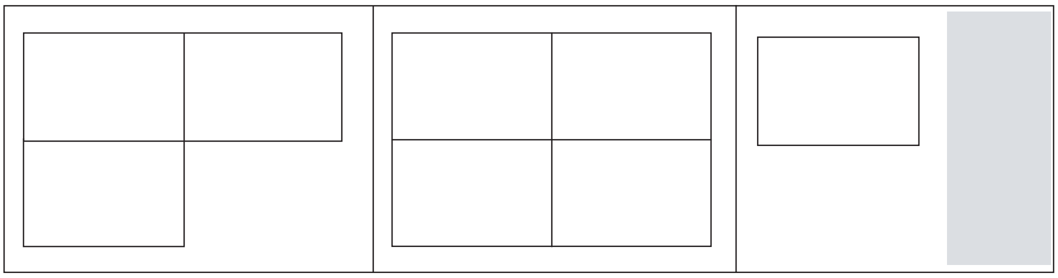


Figure 8. Secondary regions with free space.

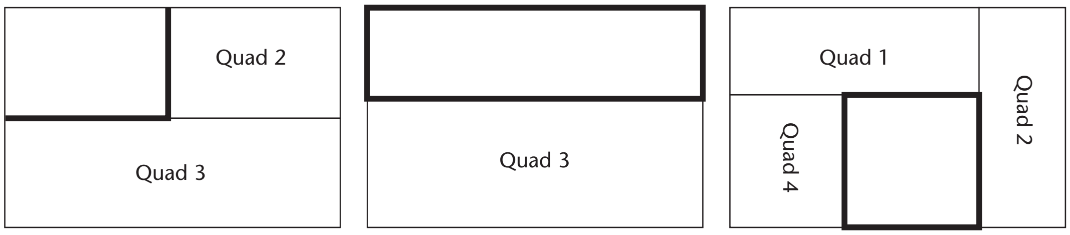


Figure 9. Edge layouts, in which some quadrants are empty.

Using the ceiling allows for columns that are only partially full, but which still take up horizontal space, as in Figure 8.

2. Because an uneven number of thumbnails exist and some regions might contain empty space, others might contain overflow. Determine the region that contains the largest number of overflowing thumbnails and keep reducing the thumbnail width for all regions by 1 pixel until there is no more overflow.

3. Redistribute whatever extra space is at the end (shaded in Figure 8) among all of the regions in the quadrant.

4. Using the current thumbnail size, repeat for the other quadrants.

5. When all quadrants have been processed, the last thumbnail size will be the correct one.

It might be possible to do Step 2 analytically, although for the pixel width and height in this context, this progressive technique is a reasonable approach.

The ReviseQuantumDim algorithm starts with the upper bound on the thumbnail width, and reduces by one each time until it reaches either a defined minimum, or zero in the worst case. If it needs to go until zero, the algorithm will run in O[*maximum thumbnail width*]. This factor will only get large in the case of a large display with few thumbnails, and even in that case it will likely terminate before the thumbnail width reaches zero.

Edge layouts. The algorithms work well for a centered rectangle. However, if the primary region is moved or sized so that a quadrant doesn't have room for any regions, edge layouts occur in which the primary region is aligned with one or more edges of the layout, as in Figure 9. To correctly handle these layouts, the RegionSplit algorithm must be modified. To do this, the system needs to determine the number of nonempty quadrants and proceed as follows:

1. If there are four, proceed to the normal RegionSplit.

2. If there are three, proceed to TriRegionSplit.

3. If there are two, perform RegionSplit only once, to yield two regions.

4. If there is one, allocate all regions to that quadrant.

5. If there are none, do not show any regions.

Each of these layout algorithms runs in time that is linear with the number of regions, as described in the initial "RegionSplit" section.

TriRegionSplit. This algorithm is a special version of RegionSplit to handle division of a collection of *n* discrete-sized pieces as evenly as possible in thirds (see Figure 10). While it's conceptually similar to RegionSplit, it's worth discussing separately, because it might not be immediately obvious how to make a bifurcating algorithm generate three regions:

1. Divide the *n* regions into two, as described in the first step of the RegionSplit algorithm. At this point, there are two regions closest to the RegionSplit split point; one to the left and one to the right, at distances a and b, where the distance is the number of thumbnails in the adjacent region.

2. Move one of the regions adjacent to the RegionSplit split point to a third pile, such that after the move, the average distance from each side to the absolute center *n*/2 is as close to equal as possible. In Figure 10, b would be added.

3. Repeat Step 2 until adding a region causes |*n*/3 – new pile's area| to be more than it was before that iteration of step 2. When this happens, backtrack one and save the result as the answer.

Postprocessing

Once the algorithm has run, certain actions can incrementally improve the automatically generated layout. We describe two here. In general, these actions can be suggested by going over the requirements of the "Problem definitions and requirements" section, and violating one of them at this later stage, whereas violating them during the initial layout would cause undesirable feedback and backtracking. For example, violating the fixed primary region size and location requirement suggests resizing or repositioning the primary region to get a "better" layout according to some metric.

Vary thumbnail size. Once the initial algorithm has run, it's possible to increment the thumbnail size in each region until any further increase would cause overflow of the region. This allows each region to have a minimum of wasted space, at the expense of the photos in different regions no longer lining up. This must be done as a postprocessing step to avoid violating the uniform thumbnail size requirement.

Add scrollbar. For the layouts to scale properly, a scrollbar can appear in a region if that region has substantially more thumbnails (currently set at 20x) than the smallest region, or than any other region, depending on the user's preference. Adding the scrollbar involves deciding how many thumbnails to show (some maximum per region), and then adding to that number on a per-region basis to enforce a full grid when not all photos are visible. For example, if a maximum of 40 thumbnails out of a region's 70 are shown, and there are seven columns, the fifth row will have only five thumbnails. In this case, two thumbnails should be added to the last row so that the grid will be full unless the scrollbar is at the last position. These must be added as a postprocessing step so that the fixed number of thumbnails requirement isn't violated during the initial layout.

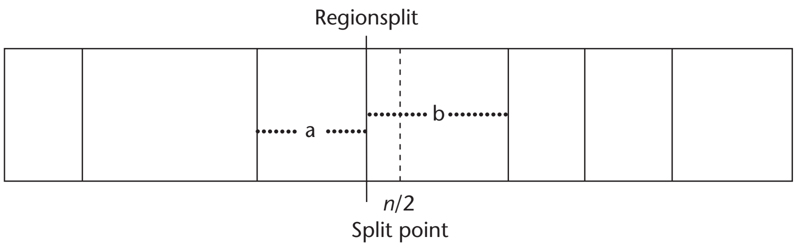


Figure 10. Dividing secondary regions among three quadrants.

Dynamic behavior

The algorithms described previously allow for interactive, dynamic behavior that encourages experimenting with primary rectangle placement, size, and overall dimensions. This goal led us to develop computed layouts, avoiding the hill-climbing or backtracking strategies that are often used in constraint satisfaction problems. The typical behavior we wished to support is to allow users to move the primary rectangle from the upper-left corner into the center, and then enlarge it to highlight its contents (see Figure 11, next page). Other behaviors include users resizing the canvas and adding and deleting regions as well as thumbnails.

Performance

To confirm the dynamic behavior described for these algorithms, we conducted three trials, in which regions were incrementally added with 10, 50, and 100 thumbnails per region, until there were 100 regions added. Execution time was measured for the three algorithms, demonstrating that the algorithms were rapid enough, with no unforeseen explosions even at the high end of 100 thumbnails in each of the 100 regions, for a total of 10,000 quanta being positioned. For this test case, the RegionSplit, InitialQuantumDim, and ReviseQuantumDim algorithms took 0.037 millisecond (ms), 0.014 ms, and 0.14 ms, respectively, on a Pentium IV

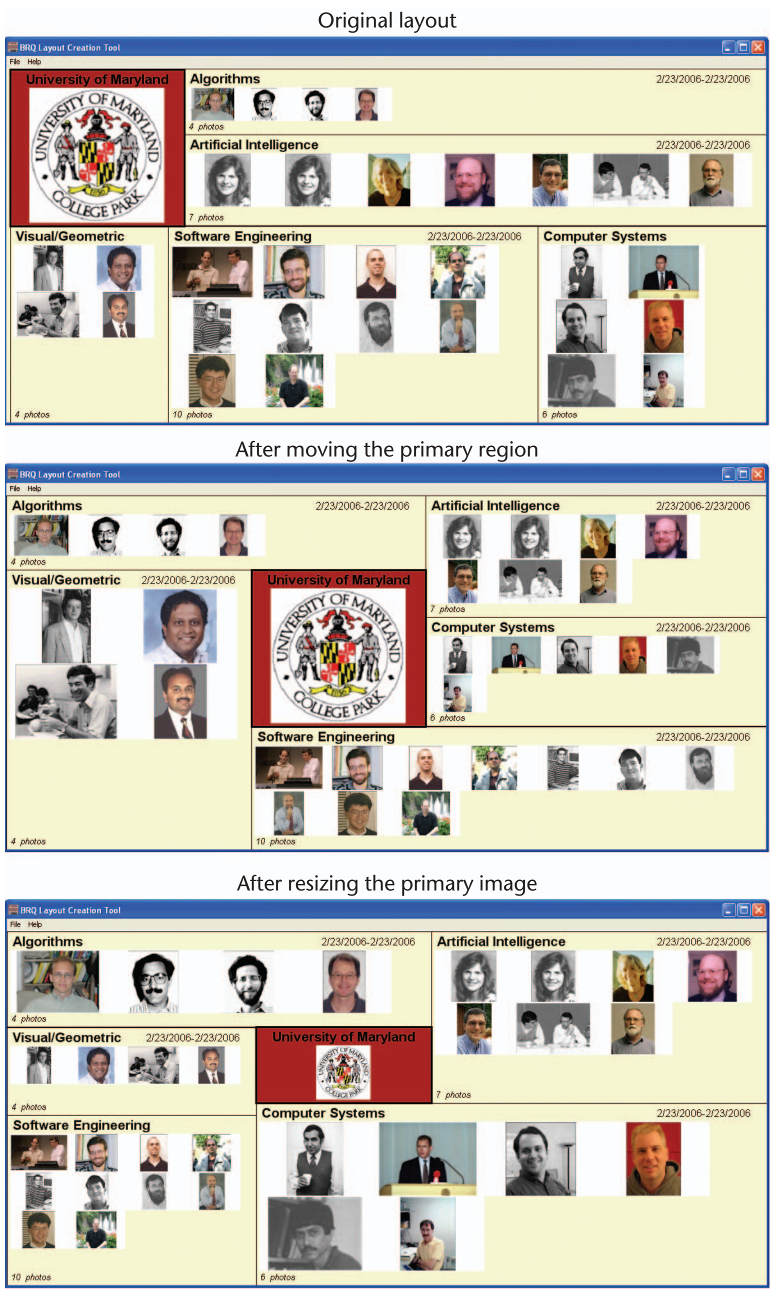


Figure 11. Dynamic behavior of BRQ layout showing user control over dragging and sizing the primary region.

2.4-gigahertz processor with 1 Gbyte of RAM. Additional tests confirmed the linear growth of time with the number of thumbnails.

Drawing the photos was more time consuming, exerting as much as three orders of magnitude (350 ms) greater drag on performance than any of the resizing algorithms. With better image storage allocation, we can solve these photo display problems, independent of the algorithmic performance outlined in this article.

Optimality discussion

It's hard to quantify what makes one photo layout better than another, as subjective measures could vary in different applications and for different users. For our purposes, however, we use the following definition to try to describe an optimal layout:

> A layout is said to be optimal if there is no layout that results in larger thumbnails.

Because any available space should be used for increasing thumbnail size, this is equivalent to a second definition:

> A layout is said to be optimal if there's a minimum of unused space.

Using these definitions, the algorithms described here produce optimal layouts given the requirements. The relaxation of one or more of these requirements could provide a layout that better meets these optimality criteria, but at the expense of ease of scanning, consistent region placement, equal thumbnail prominence, and other problems. While some of the following observations are reasonably intuitive, they're worth noting in the interest of thoroughness.

Distribution of regions into quadrants

RegionSplit divides the regions among the quadrants as evenly as possible. Imbalances tend to occur for a thumbnail distribution, which is heavy on one side (for example: five regions, where regions 1 through 4 have five thumbnails each and region 5 has 50). In this case, RegionSplit wants to put regions 1 through 4 in quadrants 1 and 2, but then region 5 would need to get placed in quadrants 3 and 4, violating one of our requirements and causing the undesirable L-shaped region. So in this example, regions 4 and 5 would go into quadrants 3 and 4, causing a reduction in thumbnail size.

Upper bound on thumbnail size

The InitialQuantumDim algorithm calculates a value for thumbnail size using height, width, and number of thumbnails, ignoring row versus column layouts and uneven distribution among regions. To show that this is an upper bound, consider w', a proposed thumbnail width that is wider than w, the value claimed to be maximal. (Since the aspect ratio is fixed, this also implies an $h' > h$.) Therefore, $w' \times h' \times$ (number of thumbnails) $> clientArea_i$, which means that the thumbnails take up more absolute area than is available.

Thus, the values *w* and *h* are upper bounds from which ReviseQuantumDim can confidently revise only downward.

Final thumbnail size

The ReviseQuantumDim algorithm will always result in at least one quadrant containing all of its rows (quadrants 2, 4) or columns (quadrants 1, 3) having at least one thumbnail, thus tightly fitting the set of thumbnails to the quadrant's client area. As a result, the other quadrants by definition have a minimum of unused space and the largest possible thumbnails, since if the thumbnails were made any larger they would overflow the tightest-fitting quadrant (because all thumbnails are required to be the same size).

Improvements by relaxing requirements

There are several ways in which users might want to improve the quality of the generated layout by manually adjusting some of the requirements. Users could choose, for example, to increase the primary rectangle's dimensions, or change its position, if an initial layout showed extra space. A manual change in ordering the regions could also result in a better layout. The "same-size thumbnails" requirement might be relaxed, with thumbnails growing to different sizes in their various regions until there is no wasted space, at the expense of having different-sized thumbnails in each region.

We should view these user actions as occurring at the application level: The algorithms discussed here provide a starting point given certain requirements, and if those are modified, they will again generate a best layout based on the new requirements. Any interaction between the application level and the algorithm layer causes feedback loops that make deterministic solutions impossible.

If the user desires a better solution, at the expense of the feedback loops previously described, an approach would be to generate a set of thumbnail dimensions for points about the current solution, for a given requirement. For example, once the algorithm finishes, it could generate what-if scenarios, varying the main rectangle's size or its position by up to some threshold. Depending on how many requirements were violated and to what extent, some number of hypothetical dimensions could be generated. Because the calculations are relatively trivial, many of these could be generated in a short period of time, and by looking at this space, a better solution could be found. This type of operation should only be allowed at the user's explicit request, as someone trying to exactly locate the primary rectangle could be frustrated by an automated agent "improving" the position to enlarge the thumbnails by changing the position that the user wants to obtain.

Usability study

To gauge user responses to these BRQ layouts we asked four knowledgeable users of photo library software to review our interface for 30 to 40 minutes.[3] In this modest usability study, they were shown the on-screen, but static, layouts in Figures 1 through 4 in order and asked what they understood about the layout and relationship among the regions in each figure. The users understood why a particular thumbnail size was chosen, although sometimes only after carefully comparing regions to find a constrained one. Several voiced interest in a feature that would relax the same thumbnail size requirement to have less wasted space. They all appreciated the ability to manipulate the layout in real time.

In response to the user feedback, we added an option to dynamically change between constant thumbnail size and allowing thumbnails to fill the available space. After that addition, users tended to choose the variable size to minimize wasted space. There were also several requests to remove photos from the layout and to choose an already-visible photo for the primary region. These were implemented with additional context menu items to allow a more intuitive direct manipulation of the final layout. One user commented that to truly generate a polished, printable finished product, full control would have to be given to the user in terms of relative positioning of text, line weights, choice of photos within a collection, and so on. While this would undoubtedly increase the tool's value, implementing these features was beyond the scope of the prototype, which was designed primarily to demonstrate the algorithms and show the potential for this type of layout.

Conclusion

Many applications, such as digital photo layouts, could use our algorithms to do dynamic layout quickly and deterministically. A vital aspect of the BRQ layout algorithm is its rapid performance, which enables compelling interactive experiences as users resize the canvas, add or delete regions, and add or delete items to a region.

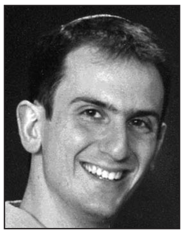
Our future work includes investigating the requirements we've described, with an eye to removing some of them, such as the fixed size and location of the primary region, or the quantum size and aspect ratio. A further challenge is to extend these ideas to a three-level hierarchical layout, which presents additional difficulties. A three-level layout of digital photos could be useful to show grandparents, parents, and grandchildren in one large ensemble, or an organizational chart with the CEO, then vice presidents, and finally senior managers in concentric rectangles. **MM**

Acknowledgments

We would like to thank Adobe Corporation for its support, which has made this research possible. We would also like to thank Ben Bederson, Alexander Loui, and Martin Wattenberg for their valuable comments; Jonathan Katz for his suggestions regarding the thumbnail sizing algorithms; and our reviewers for their detailed and thoughtful comments.

References

1. C. Jacobs et al., "Adaptive Grid-Based Document Layout," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 838-847.
2. J. Kustanowitz and B. Shneiderman,"Meaningful Presentations of Photo Libraries: Rationale and Applications of Bilevel Radial Quantum Layouts," *Proc. ACM/IEEE Joint Conf. Digital Libraries*, ACM Press, 2005, pp. 188-196.
3. S. Lok and S. Feiner, "A Survey of Automated Layout Techniques for Information Presentations," *Proc. SmartGraphics Symp.*, ACM Press, 2001, pp. 61-68; http://www.smartgraphics.org/sg01/.
4. S. Lok, S. Feiner, and G. Ngai, "Evaluation of Visual Balance for Automated Layout," *Proc. 9th Int'l Conf. Intelligent User Interfaces*, ACM Press, 2004, pp. 101-108.
5. K. Rodden and W. Basalaj, "Does Organization by Similarity Assist Image Browsing?" *Proc. Special Interest Group on Human–Computer Interaction* (SIGCHI 01), ACM Press, 2001, pp. 190-197.
6. B. Bederson, B. Shneiderman, and M. Wattenberg, "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies," *ACM Trans. Graphics*, vol. 21, no. 4, 2002, pp. 833-854.



Jack Kustanowitz is currently the director of infrastructure at the HealthCentral Network. His research interests include human–computer interaction, applications of annotated media, knowledge management systems, Internet search, and online communities. Kustanowitz has an MS in computer science from the University of Maryland at College Park, where he studied interactions between individuals and their growing collections of personal digital photos.

Ben Shneiderman is a professor in the Department of Computer Science, founding director of the Human–Computer Interaction Laboratory, and member of the Institute for Advanced Computer Studies and the Institute for Systems Research, all at the University of Maryland at College Park. He's the author of *Leonardo's Laptop: Human Needs and the New Computing Technologies*.

Readers may contact Jack Kustanowitz at jkustan@umd.edu and Ben Shneiderman at ben@cs.umd.edu.