Evaluation of Visual Balance for Automated Layout

Simon Lok, Steven Feiner, and Gary Ngai
Dept. of Computer Science
Columbia University
1214 Amsterdam Ave
New York, NY 10027
{lok,feiner,gwn2003}@cs.columbia.edu

ABSTRACT

Layout refers to the process of determining the size and position of the visual objects in an information presentation. We introduce the WeightMap, a bitmap representation of the visual weight of a presentation. In addition, we present algorithms that use WeightMaps to allow an automated layout system to evaluate the effectiveness of its layouts. Our approach is based on the concepts of visual weight and visual balance, which are fundamental to the visual arts. The objects in the layout are each assigned a visual weight, and a WeightMap is created that encodes the visual weight of the layout. Image-processing techniques, including pyramids and edge detection, are then used to efficiently analyze the WeightMap for balance. In addition, derivatives of the sums of the rows and columns are used to generate suggestions for how to improve the layout.

Categories and Subject Descriptors

H.5.2 [**HCI**]: User Interfaces—*Screen Design*; I.2.1 [**AI**]: Applications and Expert Systems; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Graphics data structures and data types*; I.4.10 [**Image Processing and Computer Vision**]: Image Representation—*Hierarchical*

General Terms

Algorithms, Design, Human Factors

Keywords

Automated Layout, Visual Balance

1. INTRODUCTION

Effective layout is one of the most important aspects of creating a presentation. By *layout,* we mean both the process of determining the position and size of each visual object that is displayed in a presentation, as well as the result of that process. We use the term *presentation* to refer to material that is intended to be viewed and manipulated by people; for example, graphical or textual user interfaces (UIs), World Wide Web documents, and even conventional newspapers and magazines.

The vast majority of layouts for information presentation today are created "by hand": a human graphic designer or "layout expert" makes most, if not all of the decisions about the position and size of the objects to be presented. Designers typically spend years learning how to create effective layouts, and may take hours or days to create even a single screen or page of a presentation. Due to the explosion in the amount of available information, there is a growing interest in automating all or part of this process.

Automated layout is a difficult problem for many reasons. Typical "computer-science" approaches, such as bin packing, tend to use the available real estate well, but fail to take into account the visual appearance or usability of a layout. Combinatorial explosion is also a problem; however, increases in processor speed have allowed the field to make some strides, since most layouts designed for visual consumption by people (e.g., as opposed to VLSI layouts) involve a relatively small number of objects.

2. RELATED WORK

The tools and techniques that an automated layout system can build upon include a wide range of approaches, such as iterative constraint solvers, formal languages for expressing layouts, and machine learning of templates, as reviewed in [15]. In addition, simpler techniques with some automated layout properties can be found in many commercial software systems (e.g., Microsoft Word, Publisher, and PowerPoint; Quark Express; and LaTeX) and in user interface toolkits (e.g., Sun JFC/Swing [13], Microsoft Foundation Classes [19], and their ancestors, such as Xtk [17] or Tk [23]).

The vast majority of the existing research on automated layout focuses on constraint systems [27, 2, 6, 9, 14, 8, 28, 22]. The idea that layouts can be presented as a set of constraints is very intuitive. One might imagine directly specifying spatial constraints, such as "Keep X above Y" and "Make X the width of the full page," or specifying abstract constraints, such as "X is related to Y" and "X is important." A system that employs abstract constraints faces the tremendous hurdle of translating the abstract notions into constraints with real spatial meaning. This extraordinarily difficult problem has been the focus of many systems to date [6, 28, 22].

Systems that employ constraint techniques often use the

IUI'04, January 13–16, 2004, Madeira, Funchal, Portugal.

concept of a design grid to design more effective layouts [5]. Design grids are a well established technique used by the visual arts community to enforce a regular pattern upon a layout that improves the visual appearance and "consumability" of the presentation [20, 10]. Recently, the concept of adaptive design grids has been demonstrated with remarkable results [11].

An alternative to constraint-based automated layout methods is to employ machine learning and demonstration-based techniques [21, 2]. These systems usually use a number of templates or "good layouts" that are generated or validated by a human designer as training sets. Some systems also incorporate a model to represent a communicative goal to provide additional parameters when creating the presentation [29].

Finally, there are evaluation techniques, the least explored of the approaches to automated layout, and the category that includes the WeightMap algorithm we present here. Existing evaluation techniques include examining the amount of mouse movement needed to interact with a presentation [26, 24, 25, 4, 12], as well as taking an information-theoretic approach to analyzing the content of a presentation [3]. An evaluation metric cannot generate a layout by itself, but it can be combined with any of the other automated layout approaches or used in a generate-and-test manner.

All previous automated layout techniques that we know of take into account visual parameters, such as balance, by leveraging input provided by a human designer (e.g., by applying templates [28]). The WeightMap algorithm is unique in that it is based on the concept of automatically evaluating visual balance without human intervention, and thus we believe that it provides a powerful new tool for automated layout.

3. VISUAL BALANCE

A graphic designer typically approaches the problem of creating a layout in a very different way from a computer scientist. Whereas the computer scientist might employ a constraint solver or use bin packing, the graphic designer starts with the concept of what "looks right" and what does not. A large part of what makes a layout "look right" is whether or not it is visually balanced. In addition to being one of the designer's fundamental measures of correctness, visual balance is also a critical aspect of the designer's workflow, as it provides the designer some idea of where and how to make modifications to the presentation.

Visual balance is a concept that is taught in virtually every first-year undergraduate course in two-dimensional design, a required course for most undergraduate degrees in visuals arts. Visual balance builds upon the notion of *visual weight,* a perceptual analog to physical weight. An object (or "form," as it is called in the visual arts world) is visually heavy if it is dark or large. In addition, if an object is textured, it appears to be heavier than if it is filled with a solid color. It is also often the case that objects closer to the center of a presentation appear visually heavier than objects at the periphery. A designer applies these rules to create effective layouts by manipulating the size and position of the objects in a presentation until the visual weights are balanced, as described below.

The established ontology prevalent in the visual arts defines three types of visual balance: symmetric, radial, and crystallographic [16]. Given a line, *symmetric balance* is the

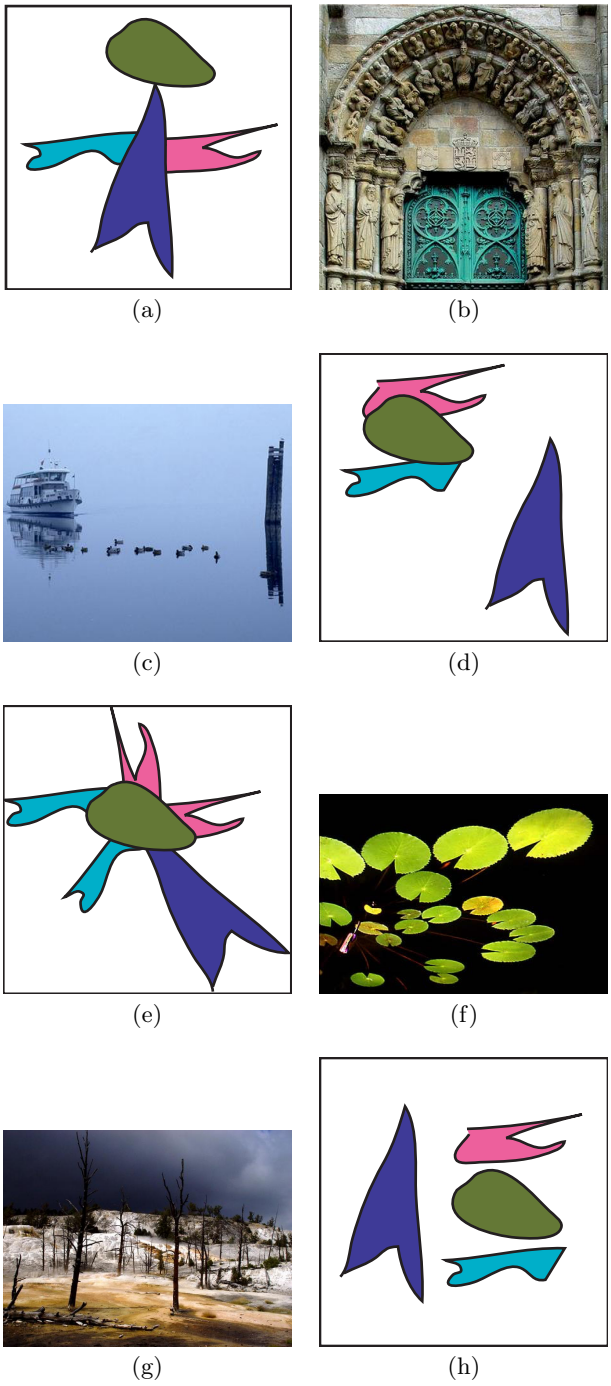
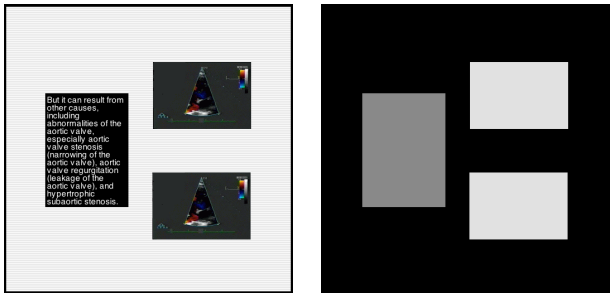


Figure 1: Examples of visual balance. (a–b) Symmetric balance. The center-line is clearly vertically oriented and runs straight down the middle of (b). (c–d) Asymmetric balance. The lighter colored, but larger, vessel at the left of (c) balances the smaller but darker mooring and shadow on the right. (e–f) Radial balance. In (f), the eye is quickly drawn to a center of attention near the lower left by the many leaves that appear to be sprouting from the bottle. (g–h) Crystallographic balance. The eye is not attracted to any particular part of (g), but rather to the overall landscape. The figure overall demonstrates asymmetric crystallographic balance.



(a) (b)

Figure 2: (a) An example layout of two pictures and a block of text. (b) A pseudocolored representation of the WeightMap of (a), where brighter colors represent heavier weights.

measure of whether the visual weight of the layout is equal on either side of the line. Some variants of this ontology make a distinction between symmetric balance, in which the actual shapes are repeated on either side of the chosen dividing line, and *asymmetric balance,* in which the visual weight is balanced, but the shapes themselves are not. Similarly, *radial balance* is the measure of the visual balance around a single point. Radial balance is typically employed to create an immediate and obvious focal point. *Crystallographic balance* is the opposite of radial balance. The idea is to visually balance the layout overall and attract the eye to the overall presentation, rather than to a single point. Figure 1 shows examples of presentations that are balanced symmetrically, asymmetrically, radially, and crystallographically.

4. WEIGHTMAP

Our approach implements the crystallographic visual balance measure. This is accomplished by creating a bitmap of the visual weight at each pixel coordinate of the presentation, which we call a *WeightMap.* In addition to being a simple measure of crystallographic visual balance, the WeightMap can be manipulated using image-processing techniques to extract suggestions about how to improve the presentation. This capability closely parallels the use of visual balance in the workflow of the graphic designer.

Let us assume that we have a display list $\vec{L}$ with n objects to display. For each object L_i, we need to have an associated visual weight W_i. For images, this visual weight can be calculated using simple pixel-processing techniques, such as taking the mean or median value of the histogram of the grayscale pixel values of the rendered L_i. In the case of text, this visual weight would be more likely to depend on the typographic parameters for the block (e.g., font, size, leading, and color). A more sophisticated measure of visual weight for a block of text would also take into account the actual characters used, and their positions. In addition, one should keep in mind that the absolute value of any W_i is not important so long as one does not cause numerical overflow. By convention, we have defined the range of W to be $0 \leq W \leq 255$, where 255 is the heaviest possible weight.

4.1 Creating WeightMaps

To create the WeightMap of $\vec{L}$, we first allocate a block of memory $\vec{M}$ equal in size to $\vec{R}$, the space to which we would normally be rendering the presentation. We then loop over each of the n elements and write the visible portions of L_i to the proper locations $M_{(p,q)}$ using standard rendering techniques. However, rather than writing the pixel value for creating the rendered display, we write the value W_i to the locations $M_{(p,q)}$. Figure 2 shows an example of a layout and its corresponding WeightMap.

Analyzing a WeightMap for crystallographic balance is very intuitive. There is a direct relationship between the uniformity of the WeightMap and the crystallographic visual balance of the layout. In the limit, a perfectly balanced layout would be one in which $\forall_{x,y} M_{(x,y)} = W_k$, where W_k is an arbitrary value. Clearly this is not feasible in the construction described thus far, because most reasonable layouts will incorporate white space (or "negative space," as it is called in the visual arts) between the objects being displayed.

4.2 WeightMap Pyramids

To address this issue, we employ the image pyramid technique from image processing [7]. Figure 3 shows an example image pyramid built from a WeightMap. By creating an image pyramid $\vec{P}$ from the weight bitmap $\vec{M}$ and dealing primarily with $\{\vec{P}_i | i \approx max(i)\}$, we work around numerous problems simultaneously.

First, the values of $\vec{P}_i$ are the average weights of the sectors in $\vec{P}_{i-1}$. Thus, white space around an object in $\vec{P}_0$ simply reduces the overall weight of the sector in $\vec{P}_i$ that an object occupies. This corresponds nicely with reality. In addition, since the pixels of $\vec{P}_i$ are averages, it is possible to satisfy the condition $\forall_{x,y} P_{i(x,y)} = W_k$, the measure of a crystallographically balanced layout described earlier.

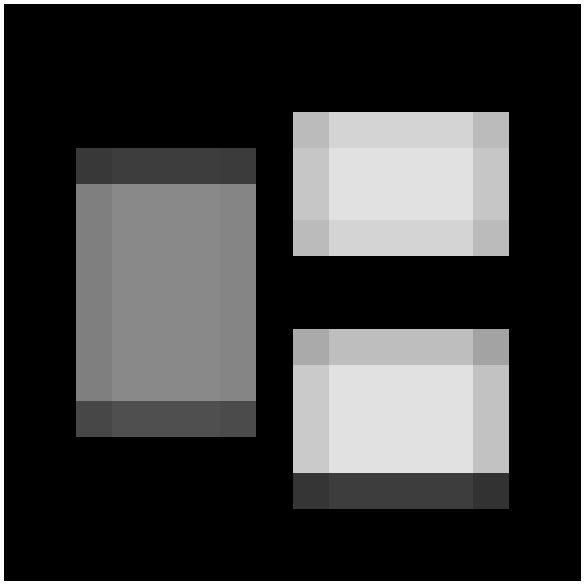
Second, since the overall size of the rendered presentation $\|R_{x,y}\|$ may be in the megapixel range, running any kind of analysis on the WeightMap $M_{x,y}$ may be too computationally intensive for a real-time automated layout system. Clearly, processing $\{\vec{P}_i | i \approx max(i)\}$ will be significantly faster than trying to analyze P_0.

By dealing primarily with $\{\vec{P}_i | i \approx max(i)\}$, we can efficiently and robustly analyze the crystallographic balance of an information presentation. The question is, what is the best value of i? If the value of i is chosen to be too large (i.e., too close to $max(i)$), then there is the possibility of the subtlety of the layout being lost due to overly aggressive averaging of the weights of objects. On the other hand, if the value is too small, then one will encounter the white space and performance issues described previously. We have found through experimentation that satisfactory results may be obtained by picking a value for i such that the pixels of $\vec{P}_i$ represent sectors of $\vec{P}_0$ that are $\approx \frac{1}{2}\|L_j\|$ where L_j is the member of $\vec{L}$ with the largest visible area in $\vec{R}$.

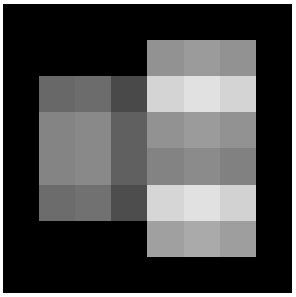
4.3 Analyzing WeightMaps

Analyzing the $\{\vec{P}_i | i \approx max(i)\}$ bitmap for uniform value is a trivial matter, both computationally and algorithmically, since $\|\{\vec{P}_i | i \approx max(i)\}\|$ will be relatively small. One can develop a number of measures of visual balance for $\vec{R}$ based on the metric $\Delta P_{i(x,y)}$ over the Euclidean product of (x, y). For example, a simple measure of overall crystallographic balance can be obtained by calculating $\Sigma_{(x,y)} \Delta P_{i(x,y)}$ and comparing that value to $\Delta P_{i(0,0)} \times \|\Delta \vec{P}_i\|$. However, getting a simple answer about the visual balance of a layout is not enough. What we want are suggestions on how to improve the layout.

One simple and efficient way to extract rules for improving



(a)



(b)

(c)

(d)

(e)

Figure 3: The highest levels of an image pyramid built out of the WeightMap of the example layout in Figure 2. (a) The fifth highest level. (b) The fourth highest level. (c) The third highest level. (d) The second level. (e) The top level. (The lower levels are not shown because they would appear to be substantially similar to the WeightMap image shown in Figure 2.)

the crystallographic visual balance of a layout is to analyze $\delta(\forall_q \Sigma_x P_{i(x,q)})$ and $\delta(\forall_p \Sigma_y P_{i(p,y)})$, the first derivatives of the row and column sums of a WeightMap. The values of these vectors can provide an idea of where to move objects within the sector of $\vec{R}$ represented by $P_{i(p,q)}$. Positive values in $\delta(\forall_q \Sigma_x P_{i(x,q)})$ suggest that the visual weight of the sector is skewed to the left, so we should move objects in that sector toward the right. Similarly, positive values in $\delta(\forall_p \Sigma_y P_{i(p,y)})$ suggest that the visual weight of the sector is skewed towards the top, so we should move objects in that sector toward the bottom.

To illustrate this, let us consider the simple case in which the display list $\vec{L}$ has a single member ($\|\vec{L}\| = 1$) and that member (L_0) is an upright rectangle. In addition, let us specify that $\|L_0\| = \frac{1}{4}\|\vec{R}\|$ and the visual weight of L_0 is W_0. Clearly, $\vec{R}$ would be visually balanced if the object L_0 were placed in the center of $\vec{R}$. In that case, the values of $\{\vec{P_i}|i \approx max(i)\}$ would be:

$$\begin{bmatrix} \frac{W_0}{4} & \frac{W_0}{4} \\ \frac{W_0}{4} & \frac{W_0}{4} \end{bmatrix}.$$

As stated previously, uniform values in $\{\vec{P_i}|i \approx max(i)\}$ is the criterion we use to determine if a layout is crystallographically balanced. Thus, the WeightMap concurs with our intuitive understanding of visual balance. Now let us place the object L_0 in the upper left corner of $\vec{R}$, so that the values of $\{\vec{P_i}|i \approx max(i)\}$ are:

$$\begin{bmatrix} W_0 & 0 \\ 0 & 0 \end{bmatrix}.$$

We know this is not a balanced layout and WeightMap concurs. In addition, since all the values of $\{\vec{P_i}|i \approx max(i)\}$ are zero except for the $P_{i(0,0)}$, the values of $\delta(\forall_q \Sigma_x P_{i(x,q)})$ and $\delta(\forall_p \Sigma_y P_{i(p,y)})$ trivially compute to W_0. According to the rules mentioned above, this tells us to move objects in the top left sector of $\vec{R}$ towards the lower right.

Clearly, if we started L_0 in the lower right corner, the values of $\delta(\forall_q \Sigma_x P_{i(x,q)})$ and $\delta(\forall_p \Sigma_y P_{i(p,y)})$ would be $(-W_0)$ which tells us to move to the upper left. A similar analysis can be performed if we start L_0 at any position and will always result in a suggestion to move L_0 towards the center.

One simple method of using these suggestions to improve the layout of $\vec{R}$ would be to move L_0 one pixel at a time in the direction suggested by WeightMap. After each iteration, one would simply repeat the process to determine the next movement. When $\{\vec{P_i}|i \approx max(i)\}$ reaches a uniform value, the layout is balanced and we can stop.

If multiple objects are present, one can generate suggestions for each of the objects and move them one at a time. In practice, we have found that it is best to move an object three or four times before manipulating the next object. In addition, it is clearly impossible for $\{\vec{P_i}|i \approx max(i)\}$ to be truly uniform in any layout other than a trivial one. Thus, it is necessary to "settle" for a best effort of close to uniformity most of the time. In practice, if the automated layout system is used in a real-time animated user interface, we have limited the algorithm to a fixed number of iterations that is small enough to maintain the desired frame rate. However, if animation is not required, the system can stop when the

values of $\{\forall_{(x,y)}\vec{P}_i(x,y)|i \approx max(i)\}$ differ by less than some constant.

Finally, we note that moving L_0 one pixel at a time is somewhat inefficient. In practice, we base the amount to move an object on a fraction of the values of $\|\delta(\forall_q \Sigma_x P_{i(x,q)})\|$ and $\|\delta(\forall_p \Sigma_y P_{i(p,y)})\|$. In general, the larger the delta, the farther one should move. However, setting the step size smaller is sometimes useful for animation in real-time user interfaces. In addition, if the step size is too large, the system can become "underdamped" and oscillate.

4.4 Complexity Analysis

Let $m = \|\vec{R}\|$, the number of pixels in the area to be managed, and $n = \|\vec{L}\|$, the number of objects in the display list. In addition, we will assume that $\forall_i \|L_i\| \leq m$. The WeightMap algorithm is articulated around four computations:

1. Determination of W_i for each object in $\vec{L}$.
2. Formulation of the base WeightMap $\vec{M}$.
3. Assembly of the WeightMap pyramid $\vec{P}$.
4. Analysis of $\{\vec{P}_i|i \approx max(i)\}$ to determine the visual balance of the layout.

Let us assume that W_i is determined by the median of the color histogram of the L_i. In this case, we need to process each pixel exactly once to place it into the appropriate histogram bucket. Thus, the time complexity of determining the weight of a single object is $\Omega(m)$, and the time complexity of determining the weight of all of the objects being displayed is $\Omega(nm)$. If we assume the objects in $\vec{L}$ will not overlap, the time complexity is $O(m)$ because we will at most have m pixels to draw. In many cases, we will not need to use the histogram approach to determine W_i. The value of W_i may be statically assigned or computable in constant time using a simple equation, especially if the object is a block of text.

To formulate the base WeightMap $\vec{M}$, we need to loop over each pixel of each object in $\vec{L}$. For each pixel, we need to do a simple comparison and potentially copy the value of W_i into $\vec{M}$. Thus the time complexity of creating $\vec{M}$ is $\Omega(nm)$. Once again, if we assume that the objects in $\vec{L}$ do not overlap, the complexity is $O(m)$.

Assembling the WeightMap pyramid $\vec{P}$ is no different than creating any other image pyramid. For each pixel at pyramid level i, one must perform 5 operations on level $i-1$. The overall time complexity for assembling the pyramid is roughly $O(m)$.

The evaluation of visual balance takes constant time c, once we have constructed our WeightMap pyramid. Since we will be using $\{\vec{P}_i|i \approx max(i)\}$ for our computations, $\|\vec{P}_i\|$ is going to be very small (typically something like 16 pixels).

Combining our results, we see that the time complexity associated with using the WeightMap algorithm to evaluate a layout is $\Omega(nm + nm + m + c) = \Omega(nm)$. However, for most layouts, we will not have overlapping objects in $\vec{L}$, so the time complexity is $O(m + m + m + c) = O(m)$. If we wish to improve the layout using the suggestions provided by WeightMap, we will need to repeatedly evaluate the layout after moving each object in $\vec{L}$. We will need to move each object at least once. In this case, the time complexity becomes $\Omega(n^2 m)$ or $O(nm)$ if we assume no overlap.

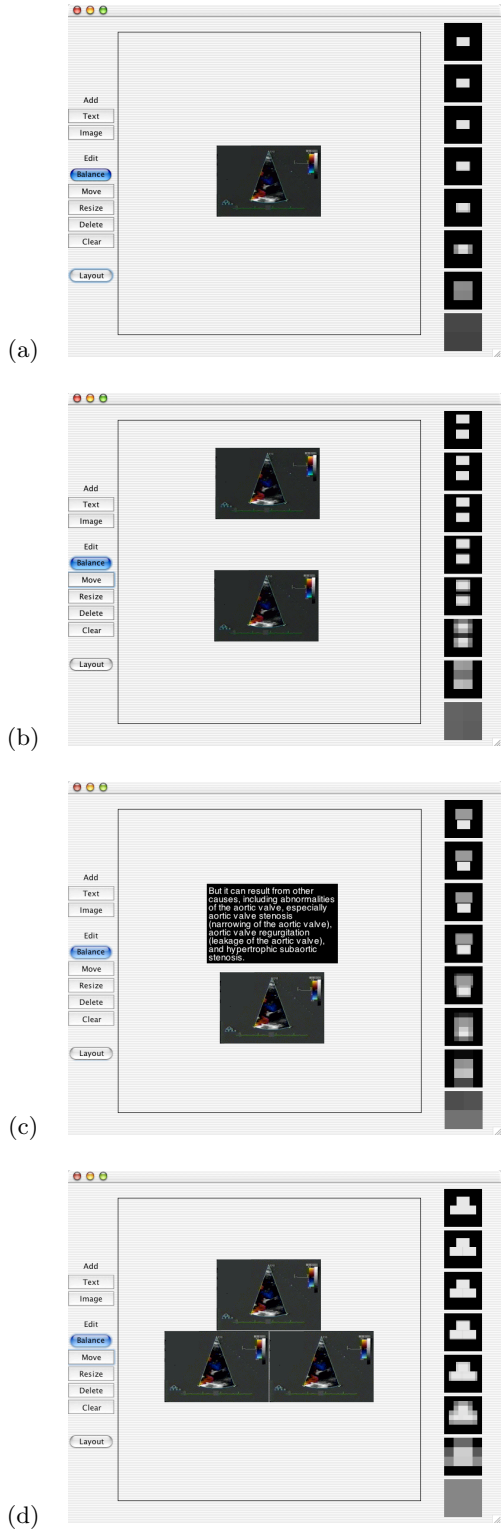


Figure 4: Screen shots of our BalanceManager test harness. The layouts shown were automatically created by BalanceManager. The initial position of the objects was the origin (the upper left corner). The test harness also displays pseudocolored uniformly-sized images of the WeightMap pyramid on the right.

The space complexity of the WeightMap algorithm is approximately $O(m)$. This is because the memory resources needed to run the WeightMap algorithm are dominated by the need to store the pyramid $\vec{P}$. Althought we also need some space to keep track of $\delta(\forall_q \Sigma_x P_{i(x,q)})$ and $\delta(\forall_p \Sigma_y P_{i(p,y)})$, these are dwarfed by $\vec{P}$ because $i \approx max(i)$, so $\|\vec{P}_i\|$ is going to be very small.

5. IMPLEMENTATION

We have implemented our WeightMap algorithm as a set of JAVA classes, allowing any JAVA-based presentation system to benefit from it. To make use of our implementation, a user instantiates a BalanceManager, passing the size of the total area to be considered in the constructor. The user then informs the BalanceManager about all the objects that are in the layout. This is accomplished by passing to the instance of BalanceManager a set of objects that implement the Box interface that we provide.

Once the BalanceManager has knowledge of all objects to consider, the user can run functions to retrieve a suggestion for a particular object under management or to execute our algorithm iteratively until a crystallographically balanced layout is reached. For the layout to be changed, the Box objects passed into the BalanceManager must be able to modify the coordinates that are used to render the objects. This can sometimes be problematic when integrating the BalanceManager with an existing application that uses its own display list. In practice, we have therefore found ourselves keeping a separate display list for the BalanceManager.

The BalanceManager uses the Cassowary constraint solver [1] to maintain spatial constraints on the layout. We have currently implemented constraints preventing Box objects from being pushed outside the bounds of the managed area, as well as preventing objects from overlapping one another.

6. RESULTS

We have developed a simple test harness program that allows a user to add, move and resize objects in a managed area, as well as query the BalanceManager for suggestions. In addition, the test harness can ask the BalanceManager to run repeatedly until completion. Figure 4 shows screenshots of our test harness and examples of the layouts it can generate. On a PowerPC 7455 1GHz processor, the test harness takes about one second to generate each of the layouts shown in Figure 4, using a stopping criterion of $\Delta\{\forall_{(x,y)} \vec{P}_i(x,y) | i \approx max(i)\} \leq 5$.

To further demonstrate the capability of the WeightMap approach, we constructed a second test harness that uses the BalanceManager to generate layouts that conform to a fixed fixed design grid, being demonstrated here with a 3×2 grid. The system first calculates the visual weight of each object by determining the median of the color histogram. Then, the BalanceManager is invoked to evaluate the balance of each of the up to $(3 \times 2)!$ possible layouts. The highest ranked layout is then displayed to the user. In the case of a tie, a random choice is made amongst the highest ranked layouts. Figure 5 shows screenshots of this second test harness and examples of the layouts it can generate.

We have also begun to integrate the BalanceManager with an experimental patient record navigation tool being developed for use at New York Presbyterian Hospital [18]. In this project, we are using the BalanceManager to control

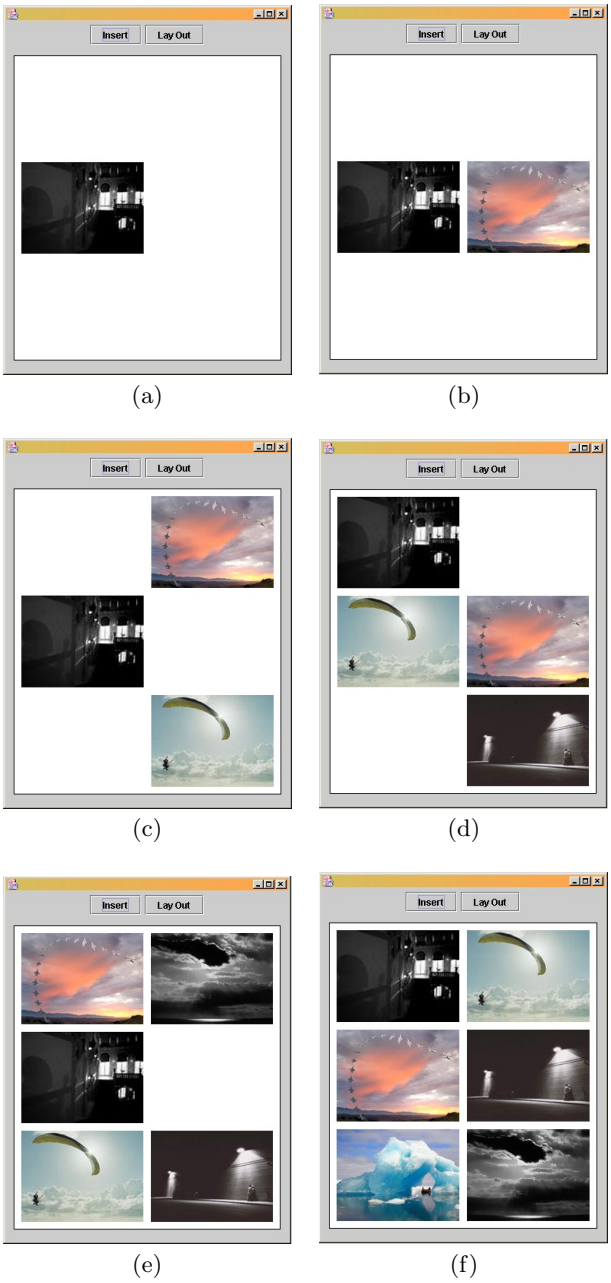
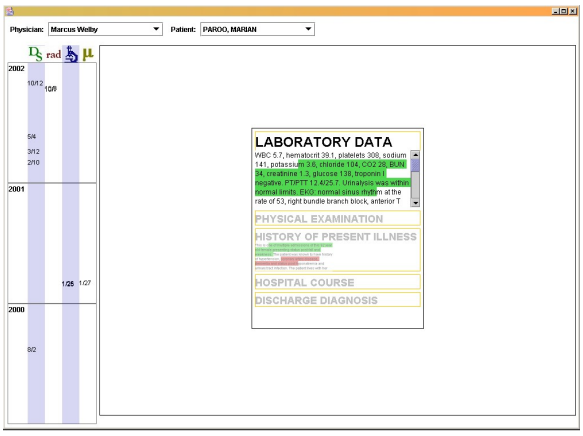
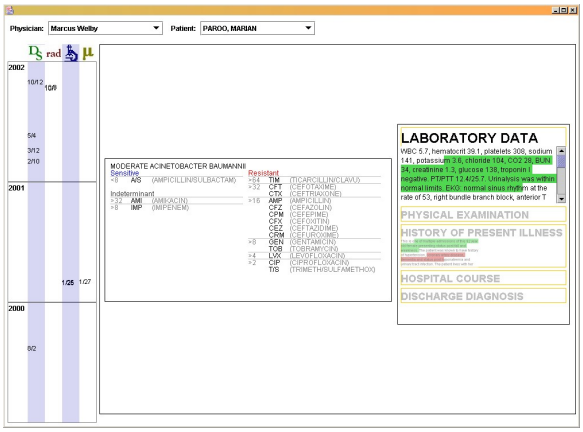


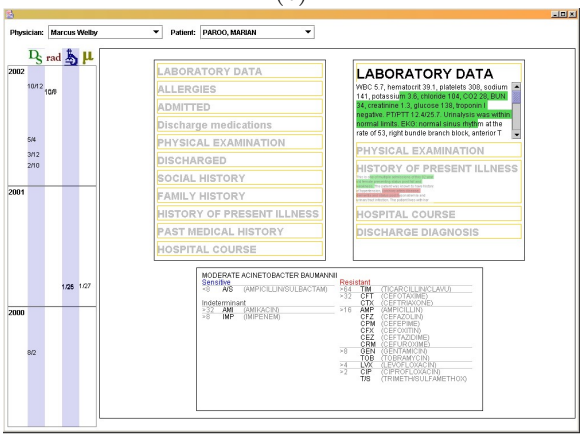
Figure 5: Screen shots of a design-grid–based test harness that show how the BalanceManager can be used in conjunction with spatial constraints. In (a) and (b), we have relatively few objects and the system has chosen to place them near the center. In (c) we see that the system has chosen to balance the two lighter pictures on the right by placing the darker one on the left. In (d), the system has chosen to balance the two darker pictures by putting them in opposite corners. (e) and (f) demonstrate how the system continues to generate balanced layouts when the layout is full of objects.



(a)



(b)



(c)

Figure 6: Screen shots of our patient record navigation user interface, as successively more objects are added to the display. The objects under the control of the BalanceManager are located in the large white area. The system initially places all objects at the origin of the managed area, which is located at the intersection of the timeline on the left and the patient and physician selector at the top. The BalanceManager automatically moves objects into the places shown in the screenshots before the screen is redrawn and presented to the user.

the placement of windows that display documents related to the patient record. Figure 6 shows screenshots from this application.

7. FUTURE WORK

We believe that the work presented here opens a large number of possible directions.

Currently, the BalanceManager does not support margins and alignment of the objects under management. To address this, we will be adding more constraints to Cassowary to support a full grid system, instead of our current approach of requiring external support for geometric constraints. In addition, we will be adding support for abstract constraints; for example, this would make it possible for related objects, such as a picture and its caption, to be kept close together. These additions would make the BalanceManager a more complete automated layout system, as opposed to its current state in which it is single tool to be used in some larger system.

We currently assume that each member of the display list has a uniform weight, and therefore use the same value for each of its pixels. We are planning to add support for per-pixel visual weight to more accurately reflect the way a human graphic designer would evaluate a layout and its elements. In addition, we will be augmenting our weight calculation approach to account for texture and smoothness by using Fourier techniques to analyze the content of the image.

We have concentrated thus far on crystallographic balance alone. We intend to expand our focus to include radial, symmetric, and asymmetric balance. For example, to address radial balance, we will experiment with overlaying a function on top of the initial WeightMap that depresses the visual weight around some point. By fixing a single object at that point, our existing algorithm will cause other objects to cluster around it.

8. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation DLI Phase 2 Initiative under award IIS-98-17434 and by gifts from Microsoft and Intel.

9. REFERENCES

[1] G. J. Badros, A. Borning, and P. J. Stuckey. The cassowary linear arithmetic constraint solving algorithm. In *ACM Transactions on Computer Human Interaction*, 2001.

[2] A. Borning and R. Duisberg. Constraint-based tools for building user interfaces. *ACM Trans. on Graphics*, 5(4):345–374, Oct. 1986.

[3] T. Comber and J. Maltby. Investigating layout complexity. In *Proc. Graphics Interface '88*, pages 192–197, June 1988.

[4] T. Comber and J. Maltby. Evaluating usability of screen design with layout complexity. In *Proc. OZCHI '95 (CHISIG Ann. Conf. on Human-Computer Interact.)*, pages 175–178, 1995.

[5] S. Feiner. A grid-based approach to automating display layout. In *Proc. Graphics Interface '88*, pages 192–197, June 1988.

[6] W. Graf. Constraint-based graphical layout of multimodal presentations. In T. Catarci, M. F.

Costabile, and S. Levialdi, editors, *Proc. AVI '92 (Advanced Visual Interfaces)*, pages 365–385. World Scientific, May 27–29 1992.

[7] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, 1986.

[8] S. E. Hudson and S. P. Mohamed. Interactive specification of flexible user interface displays. *ACM Trans. on Info. Sys.*, 8(3):269–288, July 1990.

[9] S. E. Hudson and I. Smith. Ultra-lightweight constraints. In *Proc. UIST '96 (ACM Symp. on User Interface Software and Technology)*, pages 147–155, 1996.

[10] A. Hurlburt. *The Grid*. Van Nostrand Reinhold Company, Melborne, Australia, 1978.

[11] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. Adaptive grid-based document layout. In *ACM Transactions on Graphics*, pages 838–847, July 2003.

[12] R. Jeffries, J. R. Miller, C. Wharton, and K. M. Uyeda. User interface evaluation in the real world: A comparison of four techniques. In *Proc. ACM CHI '91 Conf. on Human Factors in Comp. Sys.*, pages 119–124, 1991.

[13] Java foundation classes: Now and the future. Whitepaper, http://java.sun.com/products/jfc/whitepaper.html.

[14] S. Kochhar, J. Marks, and M. Friedell. Interaction paradigms for human-computer cooperation in graphical-object modeling. In *Proc. Graphics Interface '91*, pages 180–191, June 1991.

[15] S. Lok and S. Feiner. A survey of automated layout techniques for information presentations. In *Proc. SmartGraphics Symposium '01*, pages 61–68, Mar. 2001.

[16] B. Martinez and J. Block. *Visual Forces, an Introduction to Design*. Prentice-Hall, New York, 1998.

[17] J. McCormack, P. Asente, R. Swick, and D. Converse. *X Toolkit Intrinsics—C Language Interface*. Digital Equipment Corporation, Maynard, MA, USA, 1985.

[18] K. McKeown, S.-F. Chang, J. Cimino, S. Feiner, C. Friedman, L. Gravano, V. Hatzivassiloglou, S. Johnson, D. Jordan, J. Klavans, A. Kushniruk, V. Patel, and S. Teufel. PERSIVAL: A system for personalized search and summarization over multimedia healthcare information. In *Proc. JCDL 2001 (ACM/IEEE Joint Conference on Digital Libraries)*, pages 331–340, Roanoke, VA, June 24–28 2001.

[19] Microsoft Corp. *Microsoft Visual C++ MFC Library Reference*. Microsoft Press, Redmond, WA, 1997.

[20] J. Müller-Brockmann. *Grid Systems in Graphic Design*. Arthur Niggli Publishers, Niederteufen, Switzerland, 1981.

[21] B. A. Myers, R. G. McDaniel, and D. S. Kosbie. Marquise: Creating complete user interfaces by demonstration. In *Proc. INTERCHI '93, Human Factors in Comp. Sys.*, Apr. 1993.

[22] B. A. Myers et al. The Garnet user interface development environment. In *ACM CHI '94 Conf. Companion*, pages 457–458, 1994.

[23] J. K. Ousterhout. *Tcl and Tk Toolkit.* Addison-Wesley, 1994.

[24] A. Sears. Layout appropriateness: A metric for evaluating user interface widget layout. *IEEE Trans. on Soft. Eng.*, 19(7):707–719, July 1993.

[25] A. Sears and A. M. Lund. Creating effective user interfaces. *IEEE Software*, 14(4):21–24, July / Aug. 1997.

[26] T. S. Tullis. A computer-based tool for evaluating alphanumeric displays. In *Proc. IFIP INTERACT'84: Human-Computer Interaction*, pages 719–723, 1984.

[27] B. Vander Zanden and B. A. Myers. Automatic, look-and-feel independent dialog creation for graphical user interfaces. In *Proc. ACM CHI'90 Conf. on Human Factors in Comp. Sys.*, pages 27–34, 1990.

[28] L. Weitzman and K. Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *Proc. Second ACM Int. Conf. on Multimedia (MULTIMEDIA '94)*, pages 443–452, New York, Oct. 1994. ACM Press.

[29] M. X. Zhou and S. Ma. Toward applying machine learning to design rule acquisition for automated graphics generation. In *Proc. 2000 AAAI Spring Symp. on Smart Graphics*, pages 16–23, Stanford, CA, March 20–22 1999.