

Table of Contents

List of Figures and Tables.....	ii
Glossary	iii
Abstract.....	iv
Chapter 1: Introduction	1
1.1 <i>The Problem</i>	1
1.2 <i>Social and Ethical Implications</i>	5
Chapter 2: Literature Review	8
Chapter 3: Methods.....	13
3.1 <i>Software Development Methods</i>	13
3.2 <i>Importing Files</i>	15
3.3 <i>Generating and Manually Editing Tables</i>	16
3.4 <i>Coloring a Table</i>	17
3.5 <i>Sorting a Table by Cell Value</i>	18
3.6 <i>Matching an Image</i>	19
3.7 <i>Exporting a Table</i>	23
3.8 <i>Evaluating the software</i>	24
Chapter 4: Results	26
4.1 <i>Summary of Results</i>	26
4.2 <i>Conclusions</i>	31
4.3 <i>Recommendations</i>	32
Bibliography	35
Appendix: Source Code Listing.....	Inside Back Cover

List of Figures and Tables

Figure 1	Table components.....	1
Figure 2	Example data in list form.....	2
Figure 3	Example data in table form.....	2
Figure 4	Arbitrary table ordering illustration.....	3
Figure 5	<i>Tableware</i> displaying financial sales data.....	4
Figure 6	Financial data sorted by cell value.....	4
Figure 7	Table of financial data sorted to match an image.....	5
Figure 8	One of William Playfair's graphs.....	8
Figure 9	Technological context of <i>Tableware</i>	9
Figure 10	<i>Tableware</i> 's graphical user interface.....	14
Figure 11	The cell editor.....	16
Figure 12	HSB and RGB comparison.....	17
Figure 13	Illustration of the sorting process.....	19
Figure 14	A comparison of image sampling techniques.....	20
Figure 15	A comparison of dithering techniques.....	21
Figure 16	Initial reordering phase of the image matching algorithm.....	23
Figure 17	A table in <i>Tableware</i> and exported to <i>Microsoft Excel 2002</i>	24
Figure 18	Screenshot of <i>Tableware</i> 's usability.....	26
Figure 19	Table manipulation option dialog boxes.....	27
Figure 20	Sample data before and after sorting by cell value.....	27
Table 1	Summary of table sorting and coloring effectiveness tests.....	28
Table 2	Summary of image matching results.....	30

Glossary

<i>Ambient Displays</i>	Aesthetically pleasing displays of information which sit on the periphery of a user's attention (Mankoff et al., 2003).
<i>Change-blind</i>	Describes updates to information-displaying devices that take place when the device is not visible to the user.
<i>Compile</i>	To translate a computer program from a high-level language into another language, usually machine language, using a compiler.
<i>Data Structure</i>	Any method of organizing data on a computer for efficient manipulation.
<i>Dithering</i>	Using specific patterns of colors to give an image the appearance of having more detail.
<i>Feature</i>	An intended property or behavior of a computer program.
<i>Killer Application</i>	A computer program that is so useful or desirable that it proves the value of some underlying technology, such as a gaming console, operating system, or piece of computer hardware.
<i>Object-Oriented</i>	Characterizes a means of maintaining software quality by abstracting program structures and processes as intuitive objects.
<i>Sampling</i>	In the context of editing images, refers to a method of using the image's original pixel information to add visually consistent information to the image.
<i>Spreadsheet</i>	Table of data arranged in columns and rows often used in business and financial applications. Spreadsheet software programs are widely used computer applications that allow the user to organize large amounts of data.
<i>Usability</i>	The effectiveness, efficiency, and satisfaction with which users can achieve tasks in a particular environment of a product. High usability means a system is easy to learn and remember, efficient, visually pleasing and fun to use.
<i>Wizard</i>	A feature in a software application that walks the user through a process step by step, automating advanced tasks as much as possible.

Abstract

In the modern information age, new technologies facilitate the gathering of numerical data. These data are useless unless they can be efficiently understood. *Tableware* is visualization software that reveals trends in data by reordering and re-coloring two-dimensional tables. The software clarifies the meaning of the data by highlighting the patterns that a human can easily detect while also reorganizing the data to be visually pleasing. *Tableware* uses a set of heuristics to measure how closely the reorganized data match the user's specifications and intention for the appearance of the data, and to assess the program's effectiveness in making the data easier to understand.

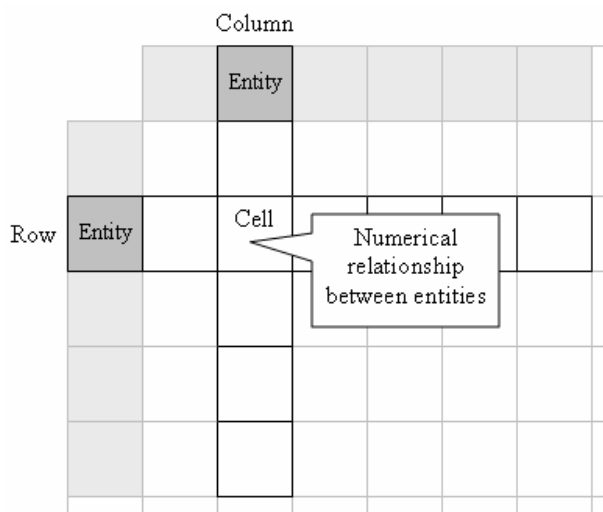
Chapter 1: Introduction

1.1 The Problem

Information is often communicated in numerical, or quantitative, form. For example, business owners must analyze the past purchases of their customer base to sell a product. Worried travelers predict future weather conditions by observing past trends on television. A student's progress is evaluated by a numerical score, and the student can compare his or her work with classmates' through statistical principals such as mean and standard deviation.

To ease communication, numerical information can be presented in ways that exploit human strengths, such visual pattern recognition. Common visualization techniques include graphs that draw numbers as points in a two-dimensional space, and charts or tables. Just as the Abstract section is a summary of this project, charts and graphs are used to simplify and summarize data.

Two-dimensional tables are a kind of graphic that represents numerical data. They have been used since the nineteenth century to convey the meaning of a set of data.



Tables consist of rows and columns such that each cell describes some relationship between two entities in the table (Figure 1). Tables are often used to present shared characteristics of a group of entities, such as the age and weight of a group of people.

Figure 1 Table components (author).

Bill is 65 and weighs 185 pounds. Sarah is 33 and weighs 130 pounds. Fred is 16 and weighs 150 pounds.
--

Figure 2 Example data in list form (author).

	Age	Weight
Bill	65	185
Sarah	33	130
Fred	16	150

Figure 3 Example data in table form (author).

Figure 2 and Figure 3 contrast a list against a two-dimensional table. Although there are no redundant data in the list, there is redundant information: the identifiers that describe each number must be repeated for each person in the data set. The table representation eliminates this redundancy and exposes quantitative relationships by establishing “age” and “weight” as characteristics shared by Bill, Sarah, and Fred.

Even though tables are more efficient and lucid than raw text, large tables can be difficult to read and modify. Computers have been programmed to create and modify tables as they are well suited to handle large amounts of quantitative data. Two-dimensional tables represented in software with accompanying computational and financial functionality are referred to as *spreadsheets* because of their paper-based counterparts. Microsoft’s *Microsoft Excel* is a commonly used spreadsheet program.

Usually, the ordering of the rows and columns of tables is arbitrary. Changing the table’s ordering does not change the information it represents. This freedom is illustrated in Figure 4; note that in both tables there are ☆△ and ☆○ relationships even though the positions of the △ and ○ columns and the ☆ row are different. Often, tables are sorted alphabetically or, if they convey events in time, chronologically to help readers find specific cells quickly.

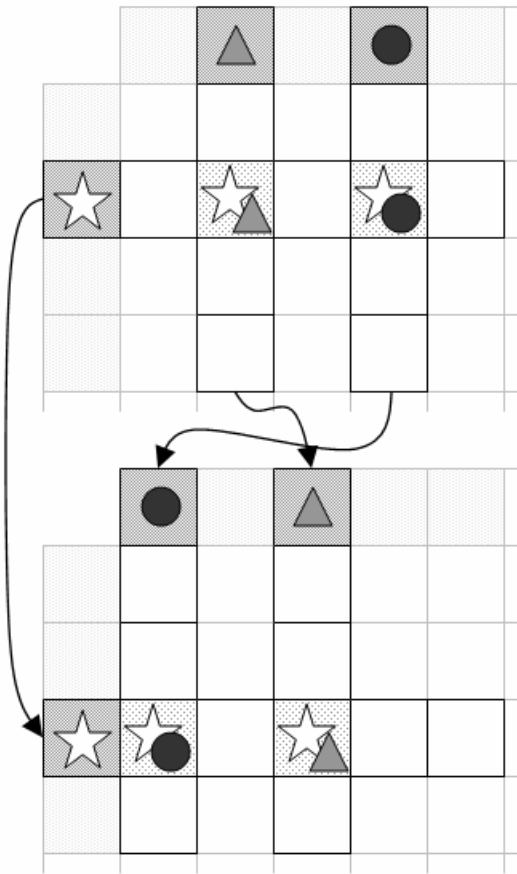


Figure 4 Arbitrary table ordering illustration (author).

Besides ordering, coloring the background of cells is another technique to visually enhance a table. A cell's color can directly correspond to its value so that readers can get a sense of a cell's value by glancing at its background color. An example coloring strategy assigns shades of red to high values to suggest those cells are "hot" and shades of blue to lower values to suggest "cold" (see Figure 5 for a table that has been colored with this strategy).

Microsoft Excel allows users to move entire rows or columns to a new position and provides background color

options as part of its cell formatting, but these features are configured manually. No automated process can intelligently reorder and recolor a table to enhance its meaning.

I have created a software application, *Tableware*, which is similar in function and appearance to *Microsoft Excel*. It exploits the inherent ordering and coloring freedom of two-dimensional tables. By intelligently editing tables, *Tableware* clarifies them and reveals trends in the underlying data while also improving the aesthetic qualities of the table. Specifically, users can either sort cells by their numerical value to cluster similar cells, or they can specify an image template and *Tableware* will manipulate the table to match the image as closely as possible.

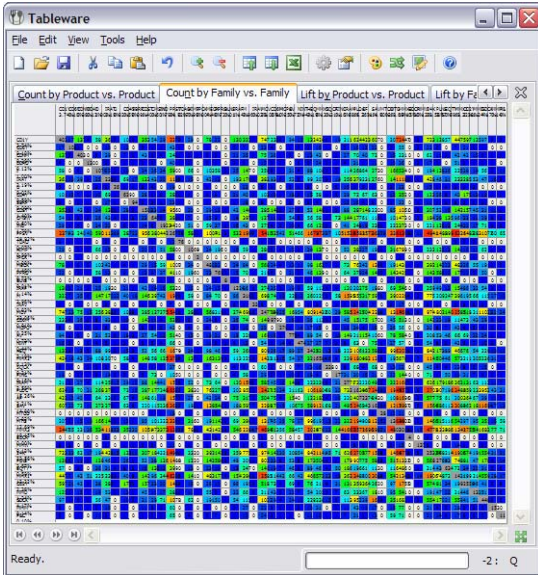


Figure 5 Tableware displaying financial sales data where each cell is the number of customers who bought both the “row” product and “column” product (author).

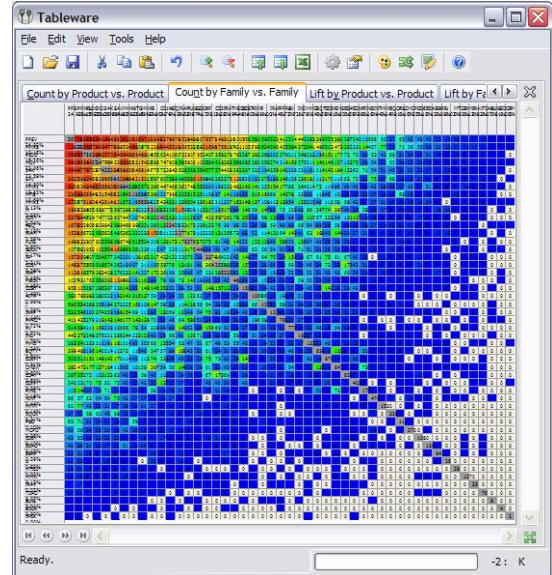


Figure 6 Financial data sorted by cell value to push products that sell well with other products to the top left corner of the table; the table has become a resource for marketing decisions (author).

Clustering similar values and shading regions of clustered cells contrasting colors visually sets the clustered cells apart from the rest of the table. This increases the readability of the table by consolidating useful information in one easily accessible location. Figure 5 shows an example table after it has been loaded and color coded into the software; Figure 6 shows the example table after a clustering sort has been applied.

Tableware can also edit a table to match an image (Figure 7). A color palette is chosen to match the color scheme of the template, and cells are rearranged so that the cells suggest the image template. This feature is less functional than sorting by cell values, but it falls into a category of visualizations called *ambient displays*. Ambient displays are aesthetically pleasing and convey information indirectly by focusing the viewer on the non-data elements of the display (Mankoff et al. 2003). *Tableware* can accurately fit a table to an image so that the user’s attention is focused on the image and not on individual cells and numbers.

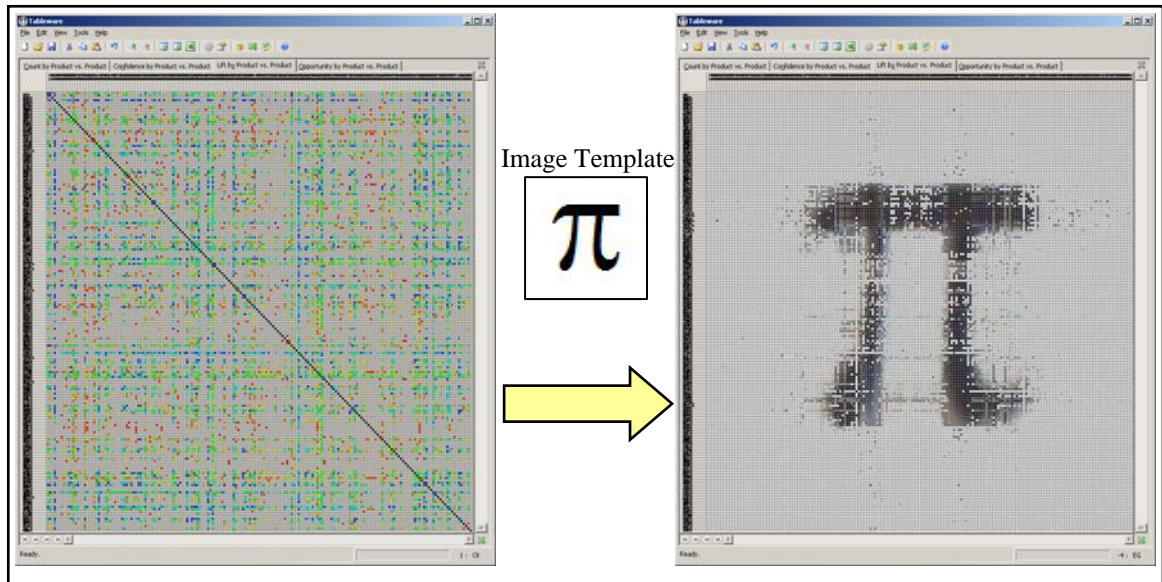


Figure 7 Table of financial data sorted to match an image of the mathematical symbol *pi* (author).

By developing *Tableware*, I have created new ways of representing data and improved on existing spreadsheet software and two-dimensional table visualization techniques. By automating table manipulation and making intelligent decisions about row and column ordering, *Tableware* decreases the time and effort needed to understand the table and increases the amount of data a human can analyze.

1.2 Social and Ethical Implications

Richard Mason identifies four areas of informational ethics: privacy, accuracy, property, and access (Mason, 1986). The first three are in fact subsets of access, since all three affect access to information.

If a visualization technique is used to display private data, the technique threatens to propagate the sensitive information fast and efficiently. Similarly, a visualization technique could make hard-to-understand or intentionally cryptic proprietary information easily understood, and thus open the door for plagiarism. Violations of privacy and copyright can facilitate illicit information access.

Information is useless unless it is accurate, so any visualization technique must not jeopardize the integrity of the data it represents for the sake of some visual effect. Edward Tufte introduced the concept of a “lie factor,” in which he was able to describe quantitatively how severely a graph or charts distorted its data (Tufte, 1983). The decisions that *Tableware* makes about how to represent data could potentially distort its true meaning. For example, a table has cells with value 0 shaded white, cells with value 100 black, and cells with 50 shaded 75% gray. This table has a lie factor of

$$\frac{\text{size of effect in graphic}}{\text{size of effect in data}} = \frac{75}{50} = 1.5$$

since cells with value 50 are too dark and thus appear to be larger. A false representation of information limits access to actual, truthful information.

Access to any source of information is unevenly distributed. Socio-economic factors and literacy affect a person’s access to the technology necessary to understand quantitative data. For example, Timothy Jenkins addresses this issue as it relates to African Americans in his article “Black Futurists.” Jenkins asserts that technology is not inherently biased to a particular social group, and that African American leadership should take advantage of societal restructuring, a product of the modern information age, to combat bias against African Americans and their access to information (Jenkins, 1997). Those who present information should ensure any tools or methods they employ work towards providing equal access to all members of their audience.

By developing new technology to display information, *Tableware* must not violate any of the areas of informational ethics. Except for cases of misuse or user error, my program saves time by providing a more efficient display of information while not jeopardizing the integrity of the represented data. I designed the software to require little

technical literacy necessary to use the functions of the program, and I designed a user-friendly interface. Since *Tableware* is a portal to information, it must be not be clogged by false data or security breaches, and access to it should be as open as possible.

Chapter 2: Literature Review

Like other software applications, *Tableware* automates a task usually performed by humans and extends existing software functionality. It therefore belongs to a hierarchy of software applications with its roots in non-electronic technology. Figure 9 provides the technological context for *Tableware*.

William Playfair pioneered pie charts, bar charts, and line graphs (Figure 8) in the eighteenth century, starting a revolution in data representation techniques; instead of representing numbers, natural patterns are now also depicted (Spence & Wainer, 1997).

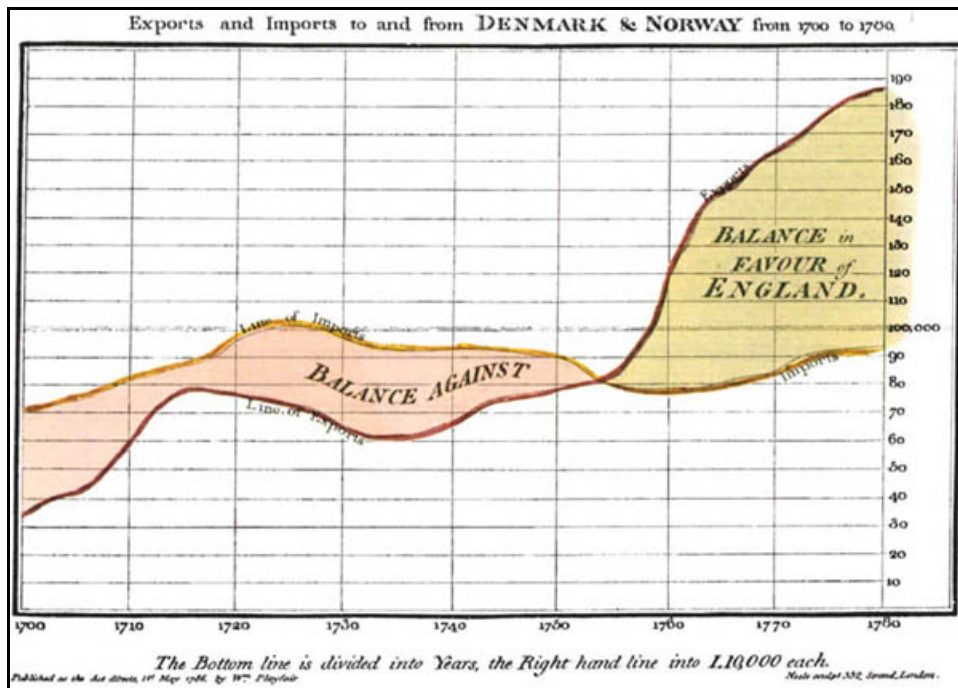


Figure 8 One of William Playfair's graphs (public domain).

Edward Tufte, an expert on data visualization, uses cognitive science to formulate design principles for graphics (Zachary, 2004). To promote “clarity, precision, and efficiency,” graphical displays should:

- Show the data
- Induce the viewer to think about substance rather than methodology

- Avoid distorting the data
- Make large data sets coherent
- Encourage viewers to compare different pieces of data
- Reveal data at several levels of detail
- Serve a purpose (Tufte, 1983).

These principles for printed graphics also apply to electronic graphs and charts. Tufte identifies four uses of color in graphs: to label, to measure (represent quantity), to represent or imitate reality, and to decorate (Tufte, 1990). *Tableware* uses color to represent a cell's value and to decorate the table.

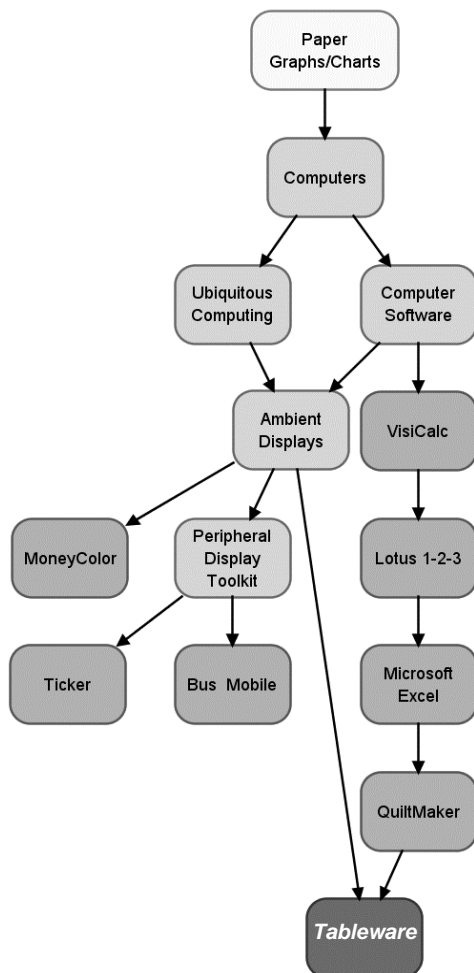


Figure 9 Technological context of *Tableware* (author).

Computers are often used to create and modify graphical displays as they are well suited to processing large amounts of quantitative data and displaying colorful graphics. Two-dimensional tables, with accompanying computational and financial functionality, form a subset of these displays and are referred to as *spreadsheets* because of their paper-based counterparts. *VisiCalc*, introduced in 1979, was the first spreadsheet program for microcomputers and was a *killer application*: software so popular that consumers purchased computers for the sole purpose of using it (Licklider, 1989).

VisiCalc, created by Harvard Business School student Dan Bricklin and MIT graduate Bob Frankston, allowed users to organize data into rows and columns, compute the results of operations applied to each row and column, and format the spreadsheet to clarify the data's meaning (Power, 2004).

A product manager for *VisiCalc*, Mitch Kapor, programmed the next-generation spreadsheet program *Lotus 1-2-3*. It offered graphing features and minimal database functionality. The second version of *Lotus 1-2-3* also offered macros, which allowed the user to store and repeatedly execute a series of commands (Power, 2004). This was an early step towards automating tedious spreadsheet tasks.

Microsoft Excel, released in 1985, featured a rich graphical user interface; the intuitive point-and-click interface model contributed to *Excel*'s popularity as it further bridged the gap between numerical data and human understanding (Power, 2004).

Microsoft Excel is now the industry standard in spreadsheet software and is therefore commonly used to visualize two-dimensional tables (O'Brien, 2006).

Jordan Crittenden of Elder Research, Inc. developed two-dimensional table-generating software dubbed *QuiltMaker* for a specific financial purpose. John Elder of the University of Virginia's Systems department saw the opportunity to clarify the meaning of the generated tables, or "quilts," through re-coloring and reordering table cells; thus, *QuiltMaker* is the inspiration for *Tableware*. Unlike *Microsoft Excel*, where the user manually configures cell coloring, *QuiltMaker* automatically shades the background of each cell according to its value. However, the row and column ordering of quilts is immutable, unlike *Tableware*.

Tableware is not an exclusive descendant of spreadsheet software. Mark Weiser introduced the idea of ubiquitous computing, where computers are integrated seamlessly into the daily tasks of life such that they present information while “disappearing into the background” (Weiser, 1999). An extension of this idea is the *ambient display*. Ambient displays are devices which convey information without demanding the user’s attention (Mankoff et al., 2003). Tara Matthews et al. (2003) have researched the viability of these displays in the context of Weiser’s ubiquitous computing, and have produced the *Peripheral Display Toolkit* to aid developers in producing ambient displays.

Examples of ambient displays developed using the *Peripheral Display Toolkit* include the *Bus Mobile* and the *Stock-News Ticker*. The *Bus Mobile* indicates to a college student how soon a bus will arrive at a nearby bus stop. The mobile hangs inside the student’s dormitory and uses symbols attached to string representing each bus route. As a bus approaches, the corresponding string and dangling symbol is retracted into the device. The mobile becomes a kind of decoration and does not require a user’s attention to convey information about the bus schedule (Mankoff et al., 2003). The *Stock-News Ticker* displays top headlines and popular stock data, but only updates its display when the ticker is not visible. This update strategy is described as *change-blind* since the user is not aware of when the ticker updates its information and is not interrupted by ticker updates (Matthews et al., 2003).

Both the *Bus Mobile* and the *Stock-News Ticker* are animated ambient displays, since they update the information they display in real time. Alternatively, ambient displays can convey information statically. An example of an ambient display that is both static and animated is the *MoneyColor* device (Shen & Eades, 2005). *MoneyColor*

generates landscape images and adjusts the landscape features (such as sky color and the number of mountains and trees) according to a stock's performance. The images are static, but they are updated as new stock information is received.

Tableware is a static ambient display. The user can sort a table to match an image template. When a reader looks at the table, he or she sees the image. Although the table's information is present, it is on the periphery of the user's attention. Functionally, *Tableware* is a descendent of spreadsheet software, but aesthetically, it is a kind of ambient display.

Chapter 3: Methods

3.1 Software Development Methods

The goal of this project was to develop algorithms for manipulating two-dimensional tables. Algorithms must be implemented to be applied to a problem. I developed a software application to implement my table manipulation algorithms.

Tableware is written in the Java programming language. Java is an *object-oriented* programming language, and has many features to aid software development. A developer can use an Integrated Development Environment (IDE) to edit, *compile*, and execute source code. I used Eclipse, a free IDE developed by the same company that created and maintains Java.

Software applications require an interface to receive input from and display output to the user. I chose to create a graphical user interface (GUI). This kind of interface is common in modern software applications and consists of components such as buttons, menus, and toolbars. Several GUI toolkits are available for Java, including Swing and the Standard Widget Toolkit (SWT). I chose to use SWT because of its flexibility and efficiency with rudimentary GUI functionality, such as loading images. Because *Tableware* is an exploratory prototype, commercial robustness is not a requirement. Therefore, the advantages of using SWT outweigh its limitations, such as only being fully supported on the Windows operating system.

To enhance the visual appearance of *Tableware*'s interface, I used icons provided by Microsoft and from a free repository (James, 2006). I also had access to Elder Research, Inc.'s interface utilities, which include source code for common tasks and interface components. Except for the table representation, I used standard interface

components (such as buttons and toolbars) to create the interface. Tables are drawn manually onto a blank area of the screen using SWT's image drawing functionality.

Figure 10 is a snapshot of the *Tableware*'s SWT interface.

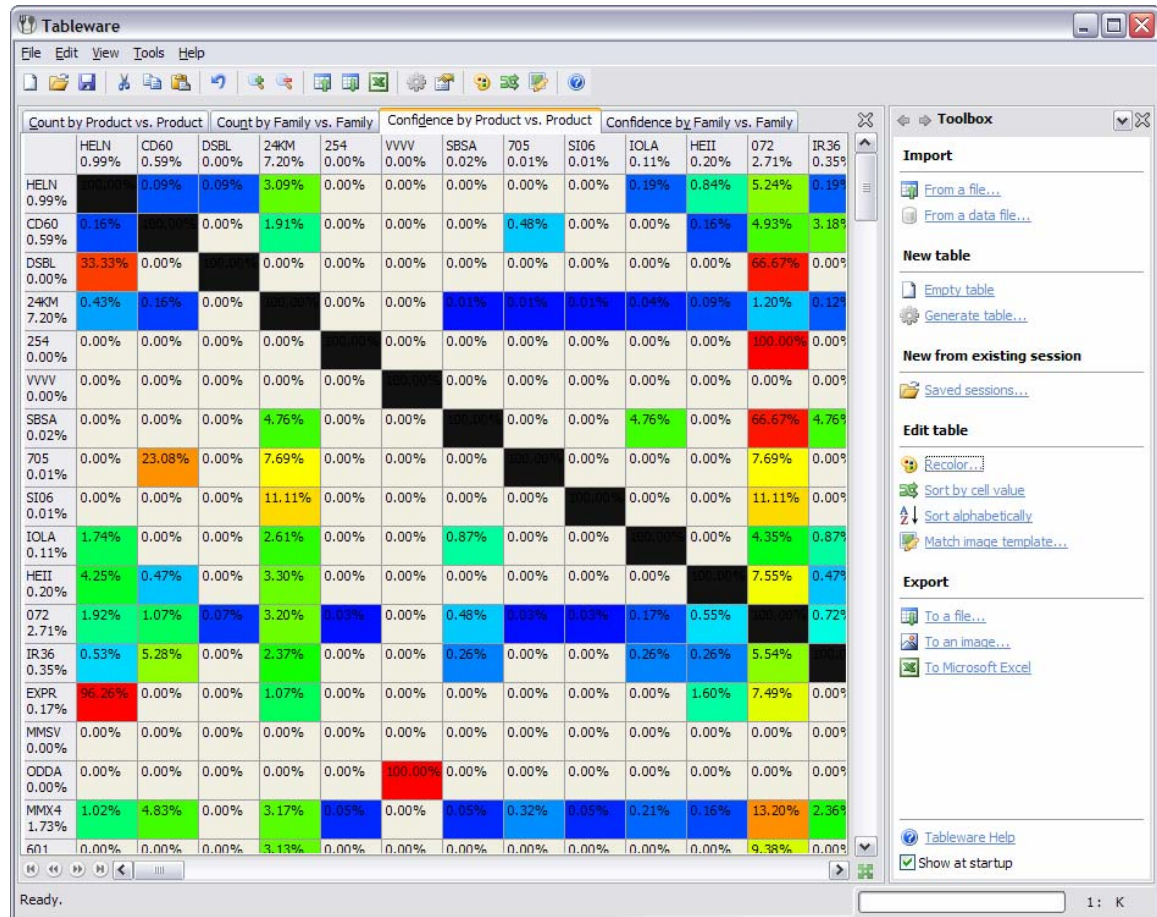


Figure 10 *Tableware*'s graphical user interface (author).

Each table's data are stored in the computer's memory and consist of:

- A matrix of floating point (decimal) numbers to store the numeric value of each cell
- A matrix of integers to store the color of each cell
- Two arrays (lists) of column and row identifying labels
- Two floating point numbers to store the lowest and highest value of the table
- A color palette that specifies how to associate colors with values

- Other display parameters, such as cell size (in pixels) and the table's name

This scheme is the optimal way to represent a table in memory while still allowing the user to edit individual cells. Alternatively, a table would not include a color matrix, but instead always calculate the color from the value when it is needed.

However, the user could not customize an individual cell's color.

A discussion of each of the software's *features* follows. Each feature provides some functionality I used while exploring and refining my table editing algorithms. While developing the software, I prioritized those features that pertained to the objectives of this project and those that contributed to the robustness and convenience of the program.

3.2 Importing Files

A user must be able to generate a table to exploit the table-editing functionality of *Tableware*. Users can load data into the program from which tables are generated. *Tableware* can import delimiter-separated text files, previously saved tables, or particular data files.

A delimiter-separated text file is a file where each row of the table is written on each line of the file, and columns are separated by the delimiter, a single character. Delimiters are often commas (in which case the file is referred to as “comma-separated values”), tabs, spaces, or semicolons. This file format is primitive and inefficient, but is easy to support using Java's built-in text file reading utilities. By supporting text files, *Tableware* can interact with *Microsoft Excel*, which can export data in this format.

The most efficient way of saving and loading tables is by leaving them in their binary form (instead of converting them to text) and saving them directly to the hard disk as they are stored in the computer's memory using Java's serialization mechanism. This

method applies only to loading saved tables, since generating files of this format is very difficult.

A descendant of Elder Research, Inc.’s *QuiltMaker*, *Tableware* can import correctly formatted data files. These data describe instances (usually business transactions) involving multiple entities (usually products). *Tableware* calculates statistical relationships between different entities and generates tables to visualize those relationships. An example statistical relationship is “confidence,” which is, in the context of retail, the probability that a customer will purchase one product given that they purchased another (Figure 10). *Tableware* reads these data files with Java’s text file reading utilities, uses *data structures* to maintain frequency information about entities in relation to each other, and generates tables using probability equations.

3.3 Generating and Manually Editing Tables

Users can also generate tables if they do not have any source data. Generated tables can be any size (as limited by the computer’s available memory) and completely blank or filled with random values. Random values can be uniformly or normally distributed (using Java’s built-in random number generator), and are constrained to a user-specified range.

With *Tableware*’s cell editing capability, users can manually set the value and color of each cell regardless of what value or color has been assigned to that cell automatically. The cell editor

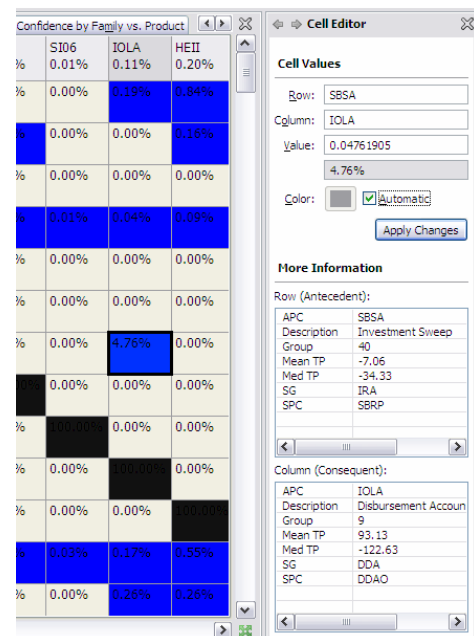


Figure 11 The cell editor (author).

also displays more information about each cell, according to text files the user has loaded. These files consist of various columns, or attributes, that describe the table's rows and columns. When loading an information file, the user specifies the column that corresponds to row and column headers, and *Tableware* populates a map from row and column header labels to the additional information loaded from each row of the file. When a cell is clicked, the cell editor accesses this mapping and refreshes its display. This feature helps the user better understand the table, where row and column labels are often numeric identifiers or cryptic abbreviations (Figure 11).

3.4 Coloring a Table

Tableware shades the background of each cell according to the cell's numerical value. Tables employ a continuous color palette where two boundary colors define the ends of the color spectrum (red for high values and blue for low values, by default). The spectrum is sampled according to the

numeric value's relationship to the table's maximum and minimum values. Colors can be defined by red, green, and blue (RGB) components or by hue, saturation,

and brightness (HSB) components (Figure 12). I chose to use a HSB spectrum for the

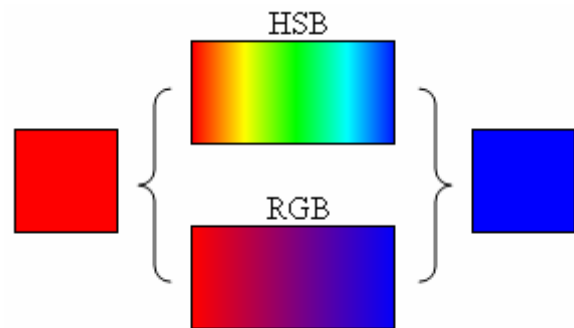


Figure 12 HSB and RGB comparison; both spectra share the same boundary colors (author).

coloring palette so that cells would have more distinct colors, and so that high values (and low values) would be more strongly emphasized. The boundary colors are customizable, and the spectrum is always a linear interpolation between them.

Tables can also be colored by rank. Instead of mapping the actual numerical value to a color, that value's rank is mapped to a color. If a cell's value one of the highest values in the table, it receives a color close to the high boundary color even though it may in fact be a very low number (the table in Figure 10 is colored by rank and includes cells with low percentages shaded a dark red). This scheme distributes the coloring more uniformly when the numbers are not uniformly distributed. When coloring by rank, *Tableware* iterates through the entire table and ranks each number, storing that rank temporarily in the color matrix. On a second pass, the rank is translated into a color.

While sorting the table to match an image, *Tableware* uses a different coloring scheme. Instead of using a continuous color palette, a discrete palette is created from the colors in the template image. Numerical values of the table are mapped to each color according to the frequency of that value in the table and the frequency of that color in the image.

3.5 Sorting a Table by Cell Value

The functional goal of this project was to make tables easier to understand. In addition to coloring cells, *Tableware* can also sort the table by numerical value. The sum of each row and column is computed and the table is sorted according to those sums. This naturally pushes higher values to the top left corner (if the sort order was descending) and lower values to the lower right corner (Figure 13). If several rows and columns have approximately the same sum, sorting by sum will appear to have no effect. To improve the algorithm's performance in this case, the ordering of the table is refined further to group similar values together. Rows and columns are shuffled, and for each

guess, a numerical score is computed to describe the extent to which similar values are grouped together. The ordering with the best score is set as the table's new ordering.

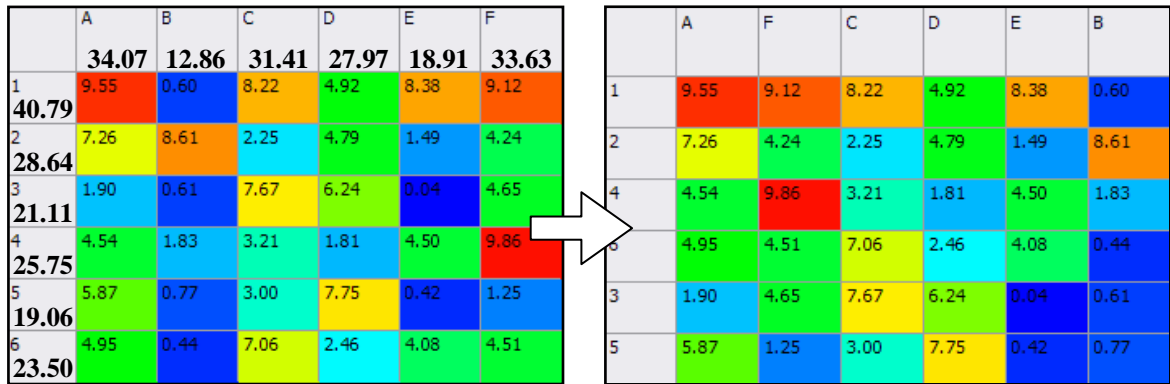


Figure 13 Illustration of the sorting process. First, the sums of each row and column are calculated (shown in bold), and then the rows and columns are sorted according to those sums in descending order. Note the proximity of cells shaded blue and red in the sorted table (author).

The computation of this score uses a prevalent digital image compression technique, the JPEG image type. JPEG images save space by storing color information for similar regions of pixels instead of storing the same amount of information for every pixel. A smaller file size for JPEG files corresponds to large regions of similar color in the image. Therefore, if the colors of a table are saved as a JPEG image, the score for that table is proportional to the file size of the corresponding JPEG image. Zach Buckner conceived this method of scoring tables using JPEG technology.

3.6 Matching an Image

The aesthetic goal of this project is to convey information subtly and in a visually pleasing way. Sorting the table so that it approximates a user-specified template image accomplishes this goal. Matching an image involves two actions: reordering the rows and columns and re-coloring the table so that the table's color palette closely matches that of the template image.

Matching a table to an image requires either the table or the image to be resized to match the dimensions of the other, since each cell in the table corresponds to a single pixel in the image. The dimensions of the table cannot be changed without losing or adding data, so the first step in the image matching process is resizing (or scaling) the template image to the same dimensions as the table. An image is composed of single points of color, or pixels, such that when an image is stretched larger than its original size, it is not obvious how to fill the gaps between the original pixels. I implemented Gaussian, bilinear, and nearest pixel *sampling* to resize the image for visually consistency with the source image (Figure 14).

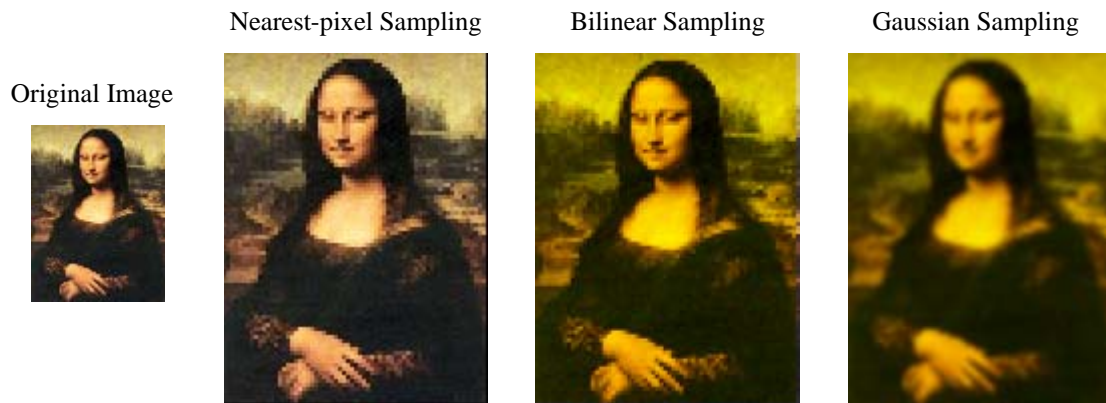


Figure 14 A comparison of image sampling techniques (author).

Since tables represent decimal numbers, each cell's value could be one of a very large number of values (theoretically, an infinite number of values, but only a finite number of values can be represented on a computer). Also, each pixel of an image can be one of several million colors. Any mapping from values to colors would therefore consume many system resources and would be impractical to implement. I implemented Floyd-Steinberg and ordered Bayer *dithering* (Figure 15) to reduce the number of colors

in the image, which also minimizes the time and resources needed to map values to colors.

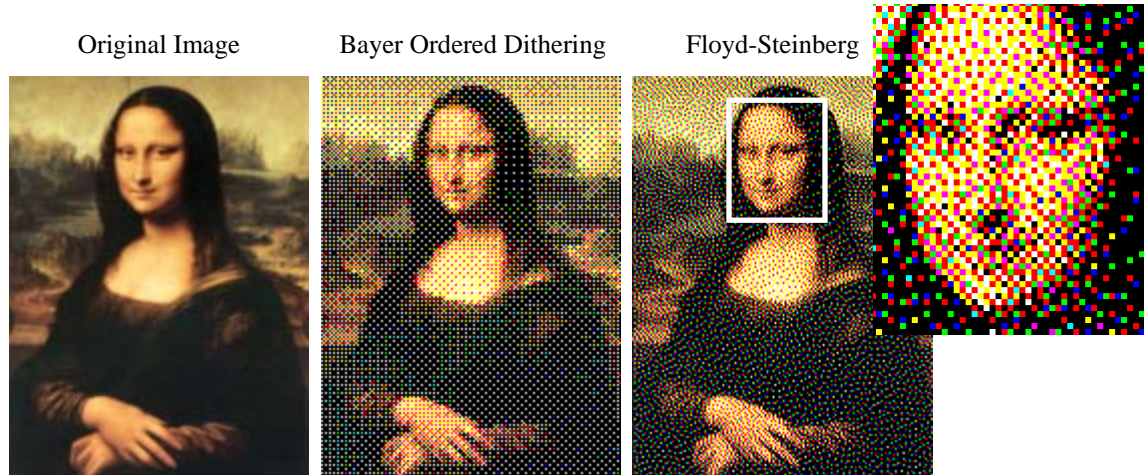


Figure 15 A comparison of dithering techniques. In this case, the number of colors composing the image has been reduced to eight, as shown in the inset (author).

Before *Tableware* attempts to reorder a table to match an image, it scales the image to the dimensions of the table, dithers the image, and then sorts the colors of the image and values of the table by frequency. Mapping values to colors based on frequency creates approximately the right number of shaded cells for each color of the target image.

The initial reordering of the table relies on the sorting-by-value algorithm described above. First, the table is sorted by cell value (Step 1 in Figure 16). A new table is created where the cell values are the colors of pixels in the image template (Step 2). The new table is also sorted by value, and the new ordering of the rows and columns is stored (Step 3). Sorting the new table by value is a means of sorting the actual image by the color of the pixels, where brighter pixels are considered higher than darker pixels. The original table is then sorted according to the order of the image template's table, but

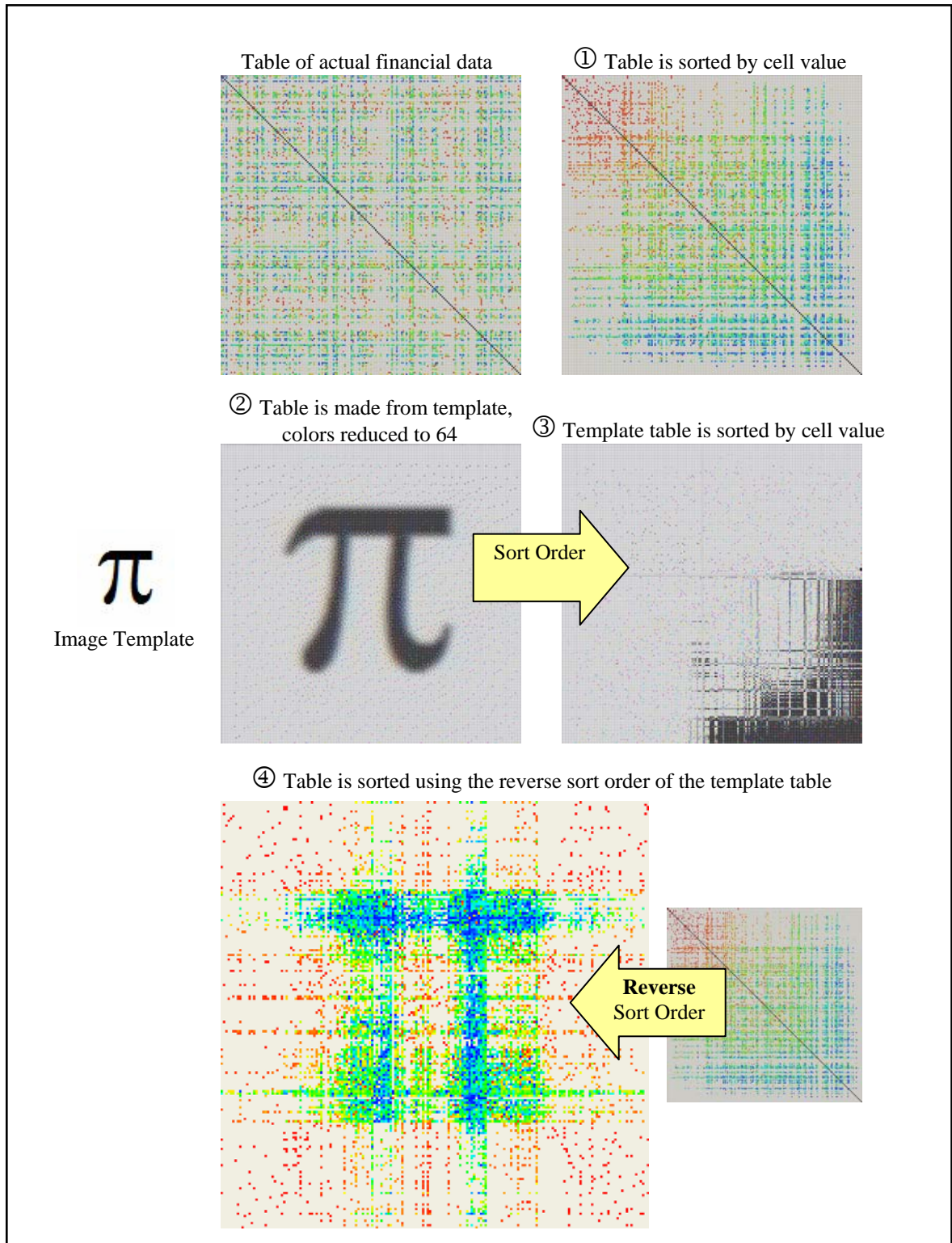


Figure 16 Initial reordering phase of the image matching algorithm (author).

in reverse (Step 4). Sorting both the table and the image by value creates a common state, and so by undoing the sort, the table and the image should be approximately equal. John Elder suggested this method of initially ordering the table to match an image.

After the initial ordering and mapping, the table has approximately the correct number of shaded cells for each color in the approximately correct positions. *Tableware* then iterates through each pixel in the target image, and maps the value of the table cell at each position to the corresponding color in the image. A value can only be mapped to one color; if more than one color should be mapped to a value, it is mapped to the average of all the candidate colors. By modifying the color of single cells, this final step captures image detail that row and column sorting overlooks. However, the accuracy of the match without the initial ordering and mapping would be poor as a value would map to a broad range of colors.

3.7 Exporting a Table

Tables can be saved to the hard disk in their binary form or as a text file (see Importing Files). Tables can also be exported as an image, or as a *Microsoft Excel* spreadsheet.

I used SWT's image writing functionality to export tables as images. The entire table as it is shown on the screen can be exported, or the color of each cell can correspond to a

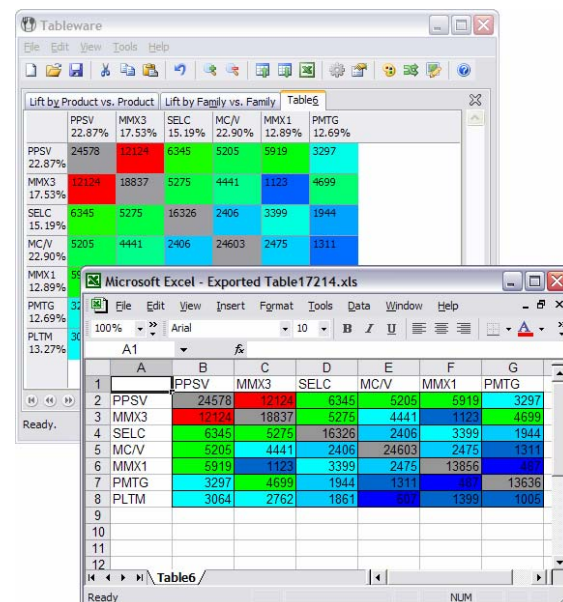


Figure 17 A table in *Tableware* and exported to *Microsoft Excel 2003*. Note the coloring is slightly different (author).

single pixel in the output image (the image shown in Step 4 of Figure 16 was generated by exporting cell colors as single pixels).

Tables can also be exported as *Microsoft Excel* spreadsheets (Figure 17). I used JXL, a Java library for writing files in the *Excel* file format (.xls). *Excel* supports shading the background of cells, but only one of 64 supported colors. When *Tableware* exports a table, it converts each cell's color to an *Excel*-supported color by finding the supported color numerically closest to the real color of the cell by computing the difference of the red, green, and blue components of the two colors. *Tableware* was designed as an enhancement to powerful spreadsheet software such as *Excel*; by providing features that interact with *Excel*, I improved my software's usefulness.

3.8 Evaluating the software

To quantitatively assess the usefulness of table sorting and coloring, I recruited five colleagues to evaluate my software. I showed participants two uncolored, unsorted tables in *Microsoft Excel* and asked them to find the largest value. I then randomly modified the maximum value, and colored and sorted the tables in *Tableware* and asked them to repeat the search. This test simulated an analysis of a business's cross product marketing potential. The data showed the number of one product that was sold in the same transaction as another product. Due to the nature of the data, diagonal cells (representing a product's correlation with itself) were uninteresting, and not candidates for the maximum value. This complicated the search and rendered most of *Excel*'s functionality (like the function `MAX`) useless. My metric for effectiveness was the time it took each participant to find the maximum value in each table. After the trials, I

interviewed each participant and asked for his feedback regarding the *usability* of my software.

The quality of an image match is heavily dependent on the nature of the data and the target image. My approach to evaluation of the matching algorithm was to try matching a variety of images to a variety of data tables. I used actual data and random values to generate tables of varying density (dense tables have an even distribution of values), and I tried matching images with differing number of color distributions (for example, the image of the π symbol is mostly white with shades of black to draw the symbol, while the image of *Mona Lisa* has a relatively even spread of browns, oranges, and yellows).

Chapter 4: Results

4.1 Summary of Results

The primary result of this project was the application *Tableware*. This software implements the table reordering algorithms I developed by providing a means for the user to create and view tables from data files. In addition to the table manipulation algorithms, I prioritized usability while developing the software (Figure 18). The interface, a main window where tables are displayed and dialog boxes that prompt the user for input (Figure 19), is very similar to *Microsoft Excel* so that users would be comfortable using *Tableware*.

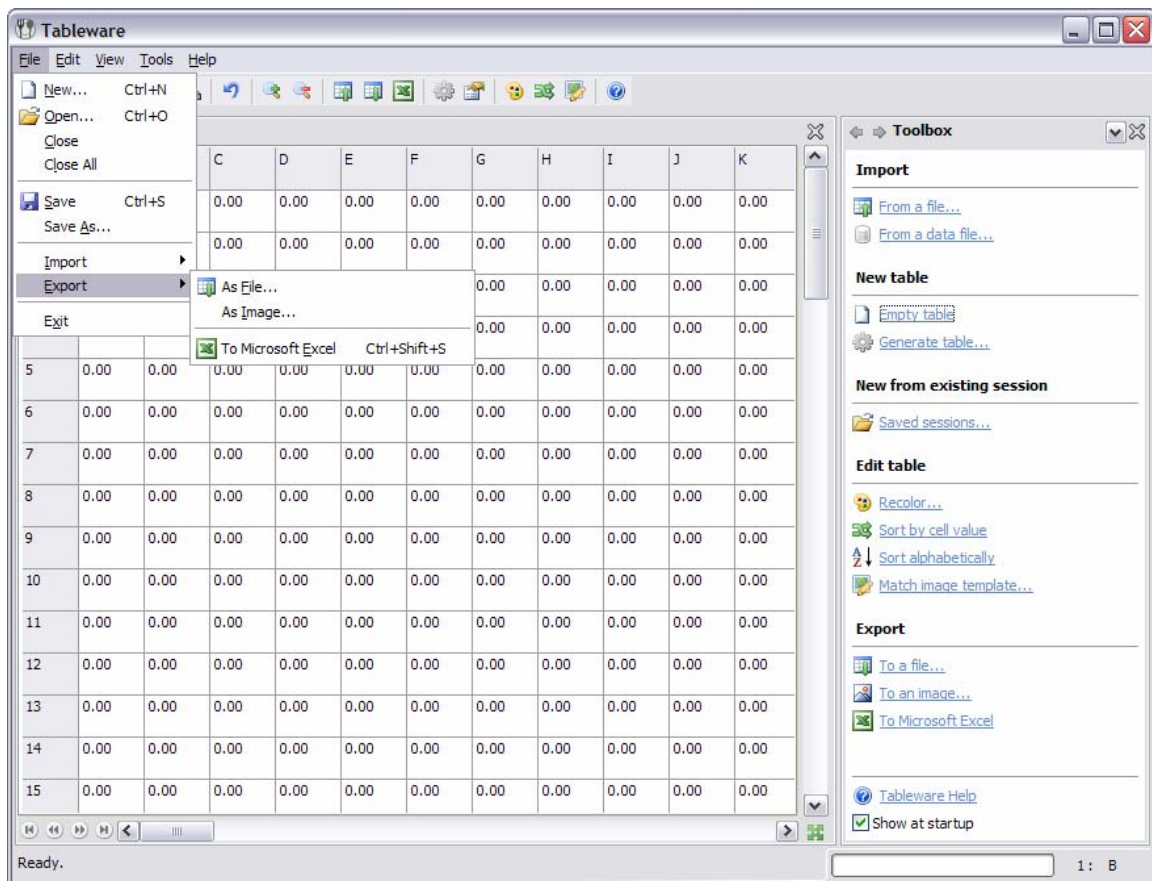


Figure 18 Screenshot of *Tableware*'s usability; commonly used features are accessible through menus, toolbars, a sidebar, and keyboard shortcuts where appropriate (author).

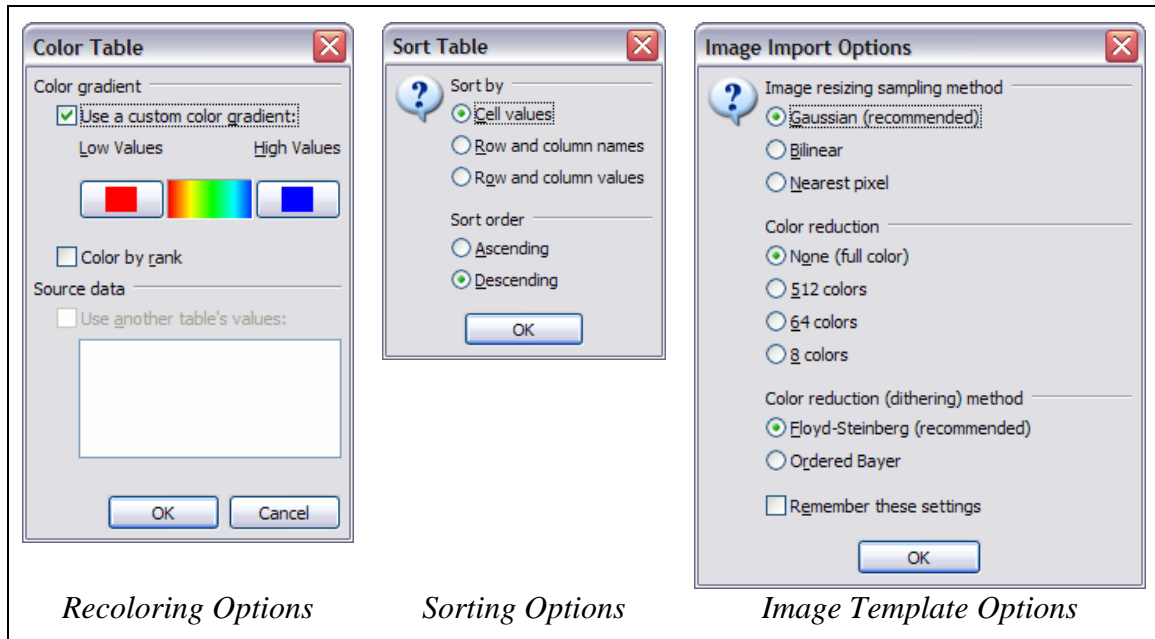


Figure 19 Table manipulation option dialog boxes (author).

My sorting algorithm successfully isolates groups of similar values (Figure 20).

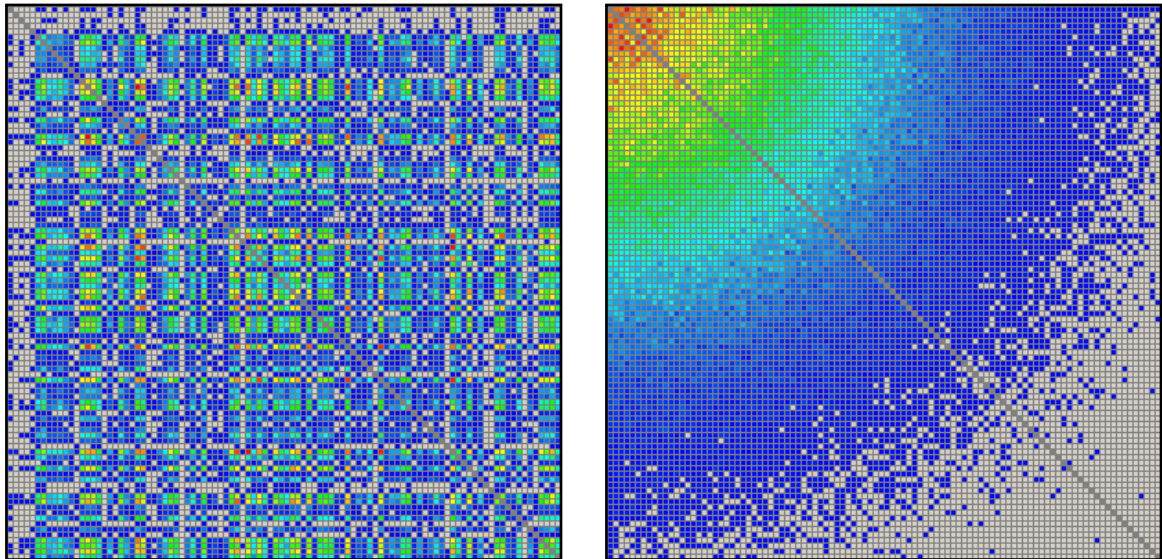


Figure 20 Sample data before (left) and after (right) sorting by cell value (author).

I evaluated the success of my sorting algorithm by asking colleagues to use my software to analyze sample data (Table 1).

Participant, Major	Table (size)	Search Time Unsorted	Ans. Right	Search Time Sorted	Ans. Right	Unsorted to sorted ratio
1 CPE/CS	1 (185 × 185)	4 m, 30 s	Yes	0 m, 28 s	Yes	9.64
	2 (380 × 380)	10 m, 0 s	No	1 m, 35 s	Yes	N/A
2 CS	1 (185 × 185)	4 m, 0 s	Yes	0 m, 42 s	Yes	5.71
	2 (380 × 380)	Gave up after 1 m		1 m, 6 s	Yes	N/A
3 ME	1 (185 × 185)	4 m, 33 s	Yes	1 m, 40 s	Yes	2.73
	2 (380 × 380)	Gave up after 2 m		1 m, 45 s	Yes	N/A
4 COMM	1 (185 × 185)	11 m, 50 s	Yes	0 m, 32 s	Yes	22.19
	2 (380 × 380)	Gave up after 10 m		2 m, 0 s	Yes	N/A
5 BME	1 (185 × 185)	4 m, 4 s	No	0 m, 8 s	Yes	N/A
	2 (380 × 380)	5 m, 35 s	Yes	1 m, 30 s	Yes	3.72

Table 1 Summary of table sorting and coloring effectiveness tests. The last column shows how many times faster the sorted search was to the unsorted search, but only when the participant found the correct answer in both cases (author).

The first participant, when asked to find the maximum value in the unsorted table, immediately began using *Excel*'s functionality to automate his task. *Excel* has the ability to sort rows based on the selected column's values and has a function to find the largest value among the selected cells. However, sorting only exposes the maximum value for one column and the `MAX()` function includes diagonal cells, so the participant attempted to use Microsoft's scripting, or "macro," mechanism. This failed as well, as the mechanism for recording his actions did not accurately repeat his single-column sort on all columns. Eventually, he visually scanned the data and answered with the incorrect answer (when analyzing the 380 by 380 table) because he had overlooked the maximum value of the table. He found *Tableware* to be efficient at finding the maximum value and the cell coloring and interface features to be particularly helpful. Due to their similar background, the second participant's experience and feedback was similar to that of the first participant.

Unlike the first two participants with similar experience using *Excel*, the third participant admitted he was "out of [his] element" when asked to analyze a two-

dimensional table. He did not attempt to make use of *Excel*'s features but instead searched manually, which he found frustrating. He also found *Tableware* more difficult to use than previous participants (although he was ultimately successful using it) and suggested I display a legend to correlate colors with cell values and provide users with a *wizard* to aid them in discovering and using the program's features.

The fourth participant, due to his background in commerce, was very familiar with the data analysis functionality of *Excel*. He eventually resorted to manually zeroing the diagonal cells and then using `MAX()` to (successfully) determine the maximum value of the sample table. This method, while tolerable for the 185 by 185 table, proved too tedious for the 380 by 380 table. He enjoyed using *Tableware* instead, except that it lacked *Excel*'s support for the mouse scroll wheel for scrolling and magnifying the table.

The final participant was the only one to successfully find the maximum value of the 300 by 300 table. He discovered a feature of *Excel* called "conditional formatting." This feature applies a specified formatting (for example, font or cell background) to all cells that meet certain criteria. He glanced at the table briefly, identified a candidate for the maximum value, and shaded cells that were greater than this value red, thus greatly reducing the search space. The majority of his search time was spent scrolling the table to make sure he had not overlooked a red cell.

To evaluate the image matching algorithm, I tried matching different kinds of tables to various template images (Table 2). Tables differed by the distribution of values, and images had varying color distributions.

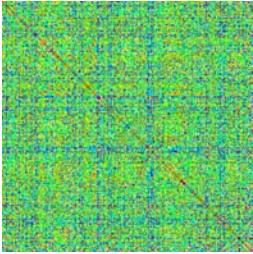
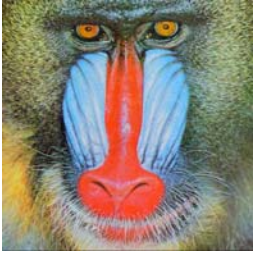
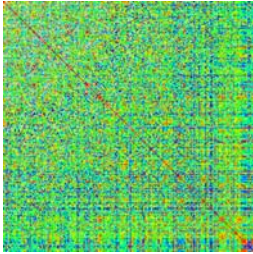

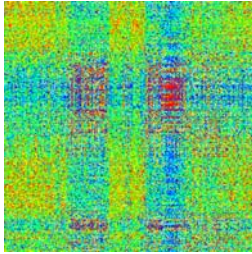

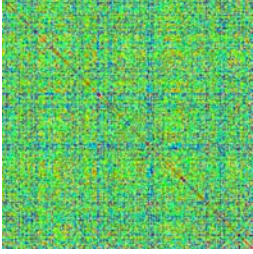

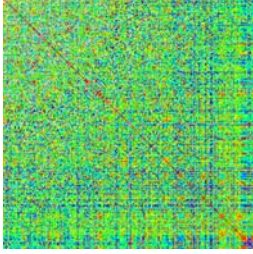
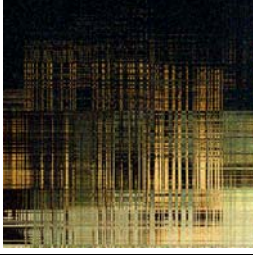
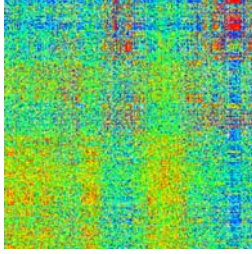

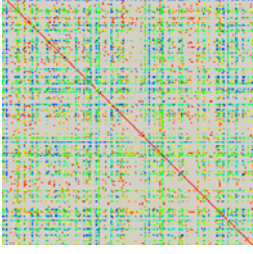
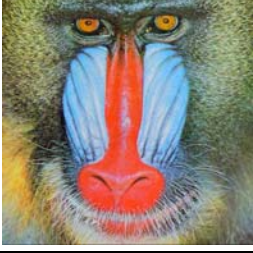
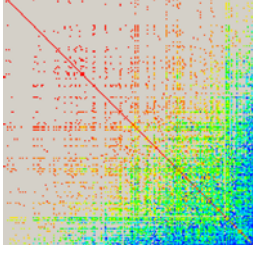
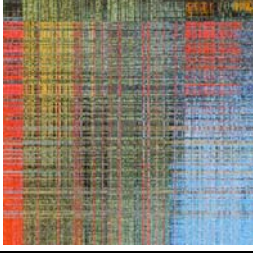
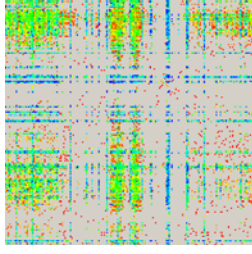
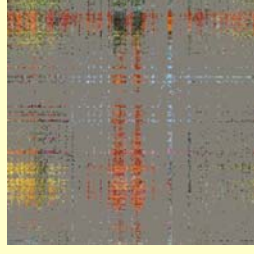
Original Table & Image	Sorted Table & Image	Reverse Sorted Table*	Final Table
 	 		
Comments The source table (actual movie rental data) was considerably dense, with most values being unique. Therefore, the image match was accurate (since most cells could map to the correct color with little conflict) and the initial reordering contributed little to the accuracy of the final match.			
 	 		
Comments Same source table as above, with a different image template. Results are similar to the previous trial, since the table is dense enough to capture the detail of the painting.			
 	 		
Comments The source table (actual financial sales data) was much sparser than the previous two trials, and thus the results for a dense image are worse. The initial reordering step clustered unique values near detail in the image, which improved the accuracy and detail of the match.			

Table 2 Summary of image matching results (author). (*Continued on next page*)

*See step 4 of Figure 16 (referred to as the “initial reordering step”)

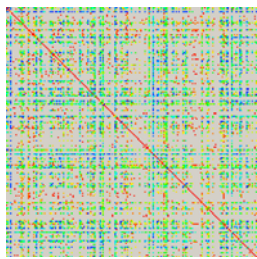
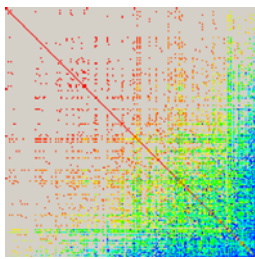
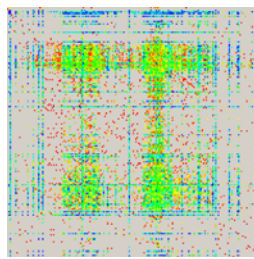
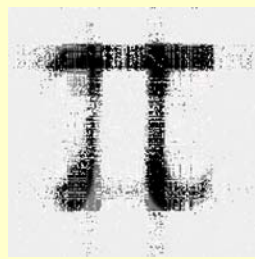
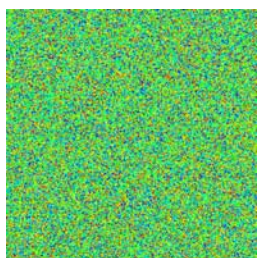
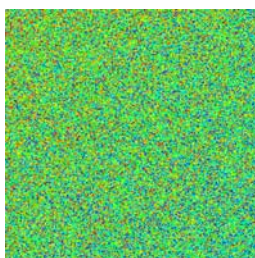
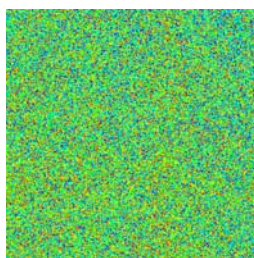

Original Table & Image	Sorted Table & Image	Reverse Sorted Table	Final Table
			
		Comments Same source table as above, with a different image template. This <i>pi</i> image is much sparser than the mandrill image, so the sparse source table is able to match it accurately. The initial reordering step and the fact that the source image is essentially monochrome contribute to the good results.	
			
		Comments As a contrast to the sparse example above, this example used a completely dense, randomly generated table. Because almost every value was unique, the table could be matched to the target image almost perfectly. The initial reordering step had a negligible effect in this case.	

Table 2 (Continued from previous page) Summary of image matching results (author).

4.2 Conclusions

The main goal of this project was to intelligently reorder two-dimensional tables; specifically, to make tables more readable through sorting and improve tables' aesthetic qualities. According to the feedback I received from my colleagues, I was successful in helping them understand large tables. My time trials showed a significant decrease in the time it took a user to analyze a table, and all participants commented on the helpfulness of shading the background of cells according to the cell's value.

The accuracy of the table matching algorithm is hard to quantify, but my results indicate I discovered a very effective algorithm. Results were poor when the distribution of the table was drastically different than that of the image. For example, if 90 percent of the cells of a table were the same value, 90 percent of the reordered and re-colored table must be the same color regardless of the detail in the image, since that value cannot be mapped to more than one color (or else the color no longer indicates the cell's value). Therefore, my matching algorithm is as accurate as possible given the constraints of the source table.

An important consideration in the achievement of my goals is the usability of the software's interface, since it directly affects the readability of tables it displays. None of the users of my software were frustrated by the look and feel of the software or the time it took to complete sorting and coloring operations, and the feedback I received was positive in general. However, participants (especially those without a computer science background) identified features that they expected *Tableware* to have, such as using the mouse wheel and the keyboard to scroll a table and providing wizards for novice users.

4.3 Recommendations

Since I implemented software, future work includes adding functionality to *Tableware* or creating new software with a similar purpose. This project explored two ways of manipulating two-dimensional tables: sorting a table by the cell values and matching an image. There are many other ways of reordering and re-coloring two-dimensional tables that future researchers could investigate.

Painting the table manually is an example of another way to manipulate tables and a possible *Tableware* feature. Cell values are mapped to colors to create a palette of

available colors. Using a paintbrush tool, the user can then select a color and shade cell backgrounds. Painting is constrained by the underlying numeric data; since colors are directly related to table data, an arbitrary number of cells cannot be shaded a particular color since the quantity of each color is dictated by the number of values that fall within that color's corresponding range of values. Also, cells cannot be arbitrarily positioned within the table, but entire rows and columns must be moved. Essentially, this feature is a hybrid of image matching and functionality offered by computer graphics programs like *Microsoft Paint*. The user creates the image template using a palette created from the table's data.

Currently, *Tableware* supports only quantitative data. The program could be extended to accept any kind of data (text, symbols, etc.) if these could be ordered, either naturally or manually by the user. For example, a table could describe some relationship between two types of animals. If the table described cross-breeding, the relationship between "Donkey" and "Horse" would not be some number, rather "Mule." *Tableware's* reordering techniques are applicable if the user established a hierarchy among cross-bred animals; if, for example, a "Liger" (the relationship, or cross-breed, of "Lion" and "Tiger") could be considered higher or bigger than "Mule," then the table could be sorted by cell value even though it does not contain numerical data.

When refining the ordering of a table to more closely match an image, *Tableware* uses an exhaustive search method that tries all possible orderings. A genetic algorithm could simplify the matching process. This class of algorithms uses criteria to assess past results to guide future guesses, much like a football coach reviews poorly executed plays from past games to improve his team's performance. To match an image or sort a table

by value, the algorithm could record attributes of and statistics about the rows and columns involved in a swap that led to a better match, and use this information to swap other rows and columns with similar attributes.

Like any software application, *Tableware* lacks exhaustive functionality for its intended use. I anticipate that new table visualization techniques will emerge as computer technology improves that will complement or exceed *Tableware*'s features.

Bibliography

- Dejoie, Roy, George C. Fowler, and David B. Paradise. 1991. *Ethical Issues in Information Systems*.
- James, Mark. 2006. Silk Icons. Retrieved February 21, 2007, from http://www.famfamfam.com/lab/icons/silk/famfamfam_silk_icons_v013.zip.
- Jenkins, Timothy & Om-Ra-Zeti, Khafra. 1997. *Black Futurists in the Information Age*. Unlimited Visions, Inc. & KMT Publications.
- Licklider, T. R. 1989. Ten years of rows and columns [spread sheet programs]. *BYTE* 14, no. 13: 324-31.
- Mankoff, Jennifer, Anind K. Dey, Gary Hsieh, Julie Kientz, Scott Lederer, and Morgan Ames. 2003. Heuristic evaluation of ambient displays. In *CHI '03: Proceedings of the SIGCHI conference on human factors in computing systems* 169-176. New York, NY, USA: ACM Press.
- Mason, Richard O. 1986. Four Ethical Issues of the Information Age. In *MIS Quarterly*. Vol. 10, No. 1.
- Matthews, Tara, Tye Rattenbury, Scott Carter, Jen Mankoff, Anind Dey. 2003. A Peripheral Display Toolkit. In *University of California, Berkeley Technotes*. UCB//CSD-03-1258.
- O'Brien, Kevin J. 2006. Microsoft wins industry standard status for Office. In *International Herald Tribune*. Retrieved February 21, 2007, from <http://www.iht.com/articles/2006/12/07/yourmoney/msft.php>.
- Power, D. J. 2004. A Brief History of Spreadsheets. In *DSSResources.com*. Retrieved February 21, 2007, from www.dssresources.com/history/sshistory.html.

- Shen, Xiaobin & Eades, Peter. 2005. Using *MoneyColor* to represent financial data. In *APVis '05: Proceedings of the 2005 Asia-pacific symposium on information visualization*. 125-129. Darlinghurst, Australia, Australia: Australian Computer Society, Inc.
- Spence, I. & Wainer, H. 1997. Who was Playfair? In *Chance*, 10, 35-37.
- Tufte, Edward R. 1990. *Envisioning Information*. Cheshire, Conn: Graphics Press, 2003 printing.
- Tufte, Edward R. 1983. *The Visual Display of Quantitative Information*. Cheshire, Conn: Graphics Press.
- Weiser, Mark. 1999. The computer for the 21st century. *SIGMOBILE Mob.Comput.Communicat.Rev.* 3, no. 3: 3-11.
- Zachary, Mark & Thralls, Charlotte. 2004. An Interview with Edward R. Tufte. In *Technical Communication Quarterly*. Vol. 13, No. 4, 447-462.