Transparent Software Methodologies in Image Processing

Steven L. Tanimoto
Dept. of Computer Science and Engineering
University of Washington
Seattle, WA 98195, USA.
tanimoto@cs.washington.edu.

Abstract

As software grows ever more complex, there is a tendency for it to become more difficult to understand. One way to help counteract these ill effects is to design software to be transparent in the sense that important structures or processes within the software are made visible. Image processing already involves data with its own natural visualizations, but there are more ways to increase transparency. Here, two particular approaches to transparency are presented. One involves showing the numeric as well as graphical representation of images, while the other involves displaying trees of image transformation relationships. The first technique is an important part of a new image processing course for undergraduates, while the second technique is aimed at users in research and development.

1 Introduction

Software systems, including those for image processing, tend to increase in power and complexity as successive products and their versions are released. While users generally welcome the increased power, the increased complexity of the software has drawbacks in terms of user understanding, and being more prone to bugs. One way to counteract these aspects is to design the software to be more "transparent."

Software systems are becoming more complex for several reasons: commercial competition, justifying upgrades to new versions, and taking advantage of new hardware or techniques. Although users may welcome the increased capabilities, the added complexity typically makes the user's task of learning the system more difficult.

By making software more transparent, that is, opening up new views of a program's computation, it is possible to alleviate, to a degree, the ill effects of software complexity.

2 Transparent Interfaces

A *transparent interface* to a software system is one that reveals some of the inner workings of the system. The inner workings are implementation details that are separate from the prescribed functional inputs and outputs of the system. Transparency is not an all-or-nothing matter, but an aspect of each datum and subprocess within the system. For each data structure and processing component of a software system, a discussion of transparency is concerned with whether to show it and how to show it. Answering the question of what to show and how to show it may involve user modeling and system modelling [3].

Transparency in a system's interface can lead to the following benefits... **Curiosity satisfied:** Curiosity is a form of cognitive tension. This tension subsides when curiosity is satisfied. **Education supported:** When a user begins using a new tool, there is typically a "teachable moment" during which the user's attention is available. Software that can present itself during this time has the potential to impart lessons of value. **Error detection/correction facilitated:** Faulty software is the norm in large commercial products. Transparency can help users detect errors. **Fluency and being "in the flow" encouraged:** Users are most productive when they can use a software system fluently. This means that they can react quickly and appropriately to changes in system state. Transparency can support this fluency by making the changes in system state more apparent. **Trust engendered:** Users may be inclined to distrust software systems for a variety of reasons including distrust of the vendor and lack of faith in the correctness of the software's functions.

Transparency can counteract such inclinations by demonstrating that the internal software state is consistent with claims made.

3 Mathematics Experiences Via Image Processing

In this section, the first of two examples of transparency in image processing is presented. It arises in a project whose goal has been to engage students in mathematics and computing by involving them in image processing activities.

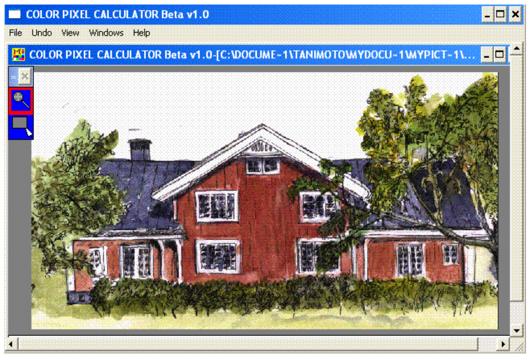
The project called "Mathematics Experiences Through Image Processing" (METIP) has been developing educational materials for image processing, mathematics, and computing at the Univ. of Washington, since 1992. It was funded by the U.S. National Science Foundation during the period 1992-97. Images appeal to students for many reasons: they are visual and the students have grown up in a visually-oriented culture of television, movies, magazine photo advertising, webpage images, and more recently, cellular-telephone photos. Mathematics, on the other hand, has been a relatively unpopular, uncool subject in school, at least in the United States. The fact that a digital image is a mathematical object thus affords an opportunity to bring students closer to mathematics.

Most image processing software hides the mathematics, because it is marketed to artists. Products like Paint Shop Pro and Adobe PhotoShop are examples of such products. In the METIP project, the opposite approach was taken, with the software showing clearly the relationship between the visual and mathematical aspects of digital images.

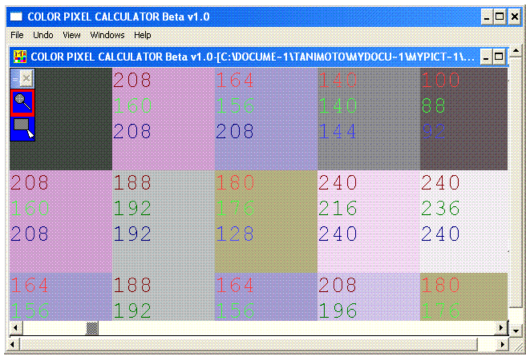
3.1 The Pixel Calculator

The "Pixel Calculator" program, developed by the METIP project, integrates visual and mathematical views of images and offers a pocket calculator-like interface with which a user can operate on images. The operations, while similar in appearance to ordinary calculator arithmetic, are actually well-defined operations in a system called "Pixel Arithmetic." For example, in Pixel Arithmetic, $255 [+] 1 = 255$. The $[+]$ operation is called "pixel addition" and it obeys "truncation semantics." The pixel operations ensure that results are in the range $\{0, 1, 2, \ldots, 255\}$ and division of pixel values is always defined, i.e., $x[/]0 = 255$, for any $x > 0$, and $0[/]0 = 1$.

In the METIP project, transparency means two things: (1) automatically showing the numeric values of pixels along with their colors whenever pixels are viewed at a suitable scale, and (2) allowing the user to employ mathematical expressions to modify images. The software does not hide the mathematics of image processing. Figure 1 shows an image of a house as displayed using the Color Pixel Calculator, (a) at normal resolution, and (b) zoomed in.


COLOR PIXEL CALCULATOR Beta v1.0
File Undo View Windows Help
COLOR PIXEL CALCULATOR Beta v1.0-[C:\DOCUME~1\TANIMOTO\MYDOCU~1\MYPICT~1\...

(a)


COLOR PIXEL CALCULATOR Beta v1.0
File Undo View Windows Help
COLOR PIXEL CALCULATOR Beta v1.0-[C:\DOCUME~1\TANIMOTO\MYDOCU~1\MYPICT~1\...

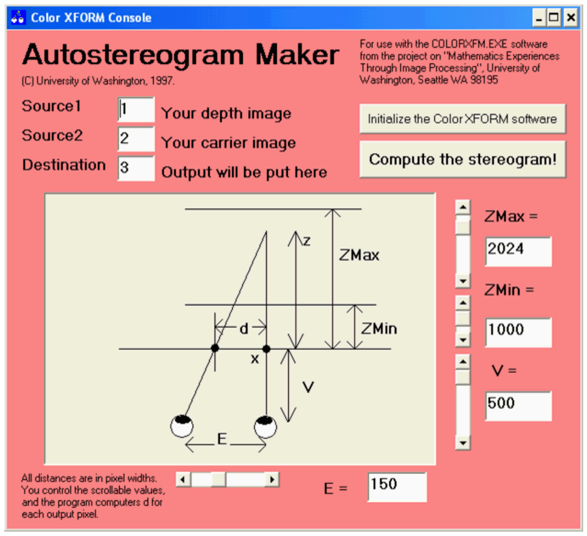
```
     208  164  140  100
     160  156  140  88
     208  208  144  92
208  188  180  240  240
160  192  176  216  236
208  192  128  240  240
164  188  164  208  180
156  192  156  196  176
```

(b)

Figure 1: Pixel Calculator displays of the house image (a) at a normal scale, and (b) zoomed-in; the RGB values are automatically displayed when the scale factor is appropriate.

3.2 Autostereograms

One of the activities used in the projec's "Pixels, Numbers & Programs" course is creating autostereograms (Magic-Eye pictures). A special interface, written in Visual Basic, to the project's "XFORM" image processing software makes the mathematics of stereogram generation apparent. This interface is illustrated in Figure 2. Of importance are the relationships among various distances, particularly the fact that there is a pair of similar triangles in the diagram; the first triangle has base E and height $V+z$; the other has base d and height z. Therefore $d/z = E/(V+z)$, which means that $d = Ez/(V+z)$. Roughly speaking, the stereogram is computed by determining the value of d at each pixel of the stereogram, and then copying the color of the pixel d units to the left (or right). Figures 3 and 4 show a conical depth map and resulting stereogram, assuming that the image of Figure 1a is used as the carrier image.


Color XFORM Console

Autostereogram Maker

(C) University of Washington, 1997.

For use with the COLORXFM.EXE software from the project on "Mathematics Experiences Through Image Processing", University of Washington, Seattle WA 98195

Source1	1	Your depth image
Source2	2	Your carrier image
Destination	3	Output will be put here

ZMax = 2024

ZMin = 1000

V = 500

E = 150

All distances are in pixel widths. You control the scrollable values, and the program computers d for each output pixel.

Figure 2: Stereogram-creation interface.

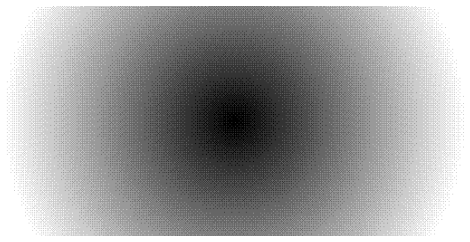


Figure 3: Depth map with the surface of a cone, computed as the distance transform of the center point.

The METIP project has developed a variety of ideas for making image processing more readily understood, and yet there remain many more techniques to be tried.

4 Strategic Transparency

While the techniques developed by the METIP project have helped to reveal the mathematical structure of images, some more recent work has striven to make visible the higher-level problem

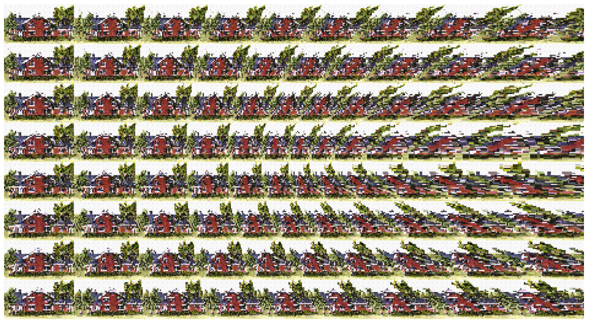


Figure 4: Stereogram created from the cone depth map and house carrier image.

solving structures used in the design of image processing experiments and applications. A system called TRAIPSE for image processing is built on a more general transparent software framework called T* (or T-STAR). Before illustrating TRAIPSE, a description of T* is given.

4.1 T-STAR

Many design and problem solving activities share a common pattern of development. One starts with a given problem and a starting arrangement or "state" (possibly simply a blank slate). Step by step, elements are either added to the arrangement, removed from the arrangement, or modified within the arrangement. For example, in designing a house, one might begin with a site map, and then one might add one room or another of the house, remove a room, modify a room, etc., until a satisfactory combination of rooms has been found. Such processes have been modeled as "state-space search" [2].

While the theory of state-space search refers to trees or more general graphs that show the relations among various possible states, the computer programs that automatically search the spaces of possibilities seldom, if ever, show these trees or graphs. As a result, a human user is not given a picture of problem-solving process and might well find the activity to be difficult to comprehend.

The T* software package provides a "Transparent STate-space search ARchitecture." It is written in Python and the GUI development kit Tkinter [1]. It provides facilities for keeping track of states in a state-space search and for displaying the tree of possibilities explored by the computer in a session. Figure 5 shows an application of T* to the well-known "Missionaries and Cannibals" puzzle. The puzzle starts with three missionaries and three cannibals and a canoe on the left bank of a river. The goal is having everybody on the right bank. Rules are (1) the canoe can carry at most three people at a time, (2) there must be at least one missionary in the boat to steer it, and (3) if the cannibals ever outnumber the missionaries either on the left bank, right bank or in the canoe, then the missionaries are eaten. The T*

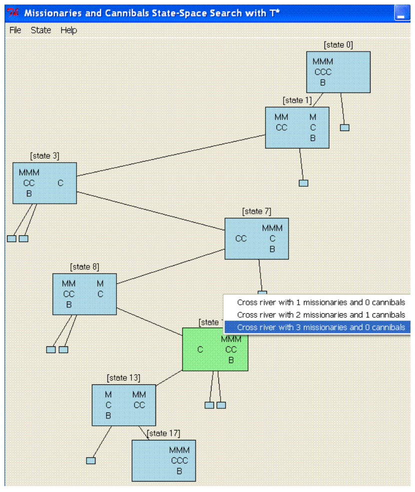

Missionaries and Cannibals State-Space Search with T*
File
State
Help
[state 0]
MMM
CCC
B
[state 1]
MM
CC
M
C
B
[state 3]
MMM
CC
B
C
[state 7]
CC
MMM
C
B
[state 8]
MM
CC
B
M
C
Cross river with 1 missionaries and 0 cannibals
Cross river with 2 missionaries and 1 cannibals
Cross river with 3 missionaries and 0 cannibals
[state
C
MMM
CC
B
[state 13]
M
C
B
MM
CC
[state 17]
MMM
CCC
B

Figure 5: T* display for a tree of explored states in the Missionaries and Cannibals puzzle.

code enforces the rules and shows the states that have been "explored" by the user.

4.2 T* in Image Processing

The TRAIPSE software adapts T* for image processing. The adaptation consists of (a) definition of image-processing states and their displays, and (b) provision of a set of image-processing operators. Each state consists of a small number of data objects, usually a string and an image. Each operator contains a precondition (a predicate on states that determines whether the operator is applicable to a given state or not), and a function that maps one state to another. TRAIPSE currently comes with only a small set of operators, mainly written as calls on the open-source Python Imaging Library. However, additional operators can be written either in Python or in C/C++ and loaded into TRAIPSE.

The transparency in TRAIPSE facilitates two types of work: (1) sequencing operators to achieve particular kinds of results, and (2) comparison of results of different operators or operator sequences. The first activity is a kind of computer programming: the formation of recipes or sequences of steps to take in order to transform given input data into useful results. The second activity is evaluation and is used in fine-tuning operator sequences.

Figure 6 shows a four-level tree with the original image at the root and various derived versions below it. The popup menu shows T* commands available for the selected node, which is highlighted in green. TRAIPSE was recently used by students

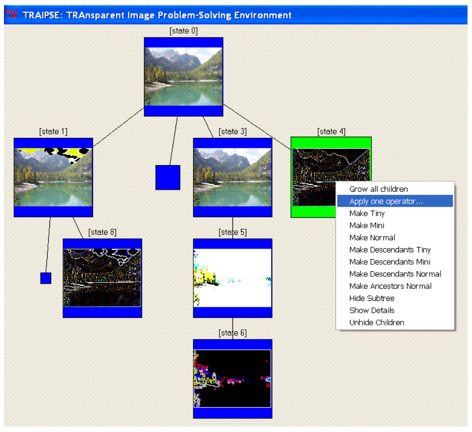


Figure 6: An example display with the TRAIPSE system.

at the University of Rome under the direction of L. Cinque for their term projects. One project team applied TRAIPSE to face recognition. Another applied it to motion analysis in image sequences.

5 Acknowledgments

For encouragement with T* thanks to S. Levialdi, L. Cinque, A. Malizia, C. Bernardelli, P. Bottoni, M. DeMarsico, and E. Panizzi. For encouraging the presentation, thanks to G. Borgefors, E. Bengtsson, I. Nyström, S. Svensson and F. Georgsson. Thanks also to the many METIP project collaborators. Finally, presentation of this paper was supported in part by NSF Grant IIS-0537322.

References

[1] Lundh, F. 1999. An Introduction to Tkinter. http://www.pythonware.com/library/tkinter /introduction/.

[2] Newell, A., and Simon, H. A. 1972. Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall.

[3] Tanimoto, S. L. 2004. Transparent interfaces: Model and methods. *Proc. Workshop on Invisible and Transparent Interfaces*, Gallipoli, Italy.