Teaching 2D Arrays Using Real-time Video Filters

Jeff Chastine
Clayton College and State University
5900 N. Lee St.
Morrow, GA
1-770-960-4309
jeffchastine@mail.clayton.edu

Jon Preston
Clayton College and State University
5900 N. Lee St.
Morrow, GA 30260
1-770-960-4354
jonpreston@mail.clayton.edu

ABSTRACT
Educators have long been trying to spice things up in their introductory programming courses. Traditionally, two-dimensional arrays have been taught non-graphically using contrived examples and the command-line – which is often not motivating to students. We believe (as does supporting literature) that the inherently visual nature of a more media-oriented approach to teaching arrays, such as teaching using Image Processing, is more effective and more engaging for students. This teaching style can be extended to include real-time video filters, opening up a unique set of time-sensitive algorithms and making the traditional image processing transformations highly interactive.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education –*Computer Science Education,* E.1 [**Data Structures**] – *Arrays*, I.4 [**Image Processing and Computer Vision**] - *Miscellaneous*

General Terms
Algorithms, Design, Experimentation.

Keywords
Real-time, Filters, Two-dimensional Arrays, Programming, Teaching

1. INTRODUCTION
Traditionally, the teaching of arrays usually involves routine array operations, leading to unimaginative assignments such as finding the average, minimum or maximum elements of the array. If the students are lucky, they may get a game-related project, like tic-tac-toe, chess, or Battleship, which partially bridges the gap between command-line and graphics through ASCII art. As noted in the paper by Egbert et al, most individuals prefer to learn visually [2]; additionally, Guzdial affirms that students involved in creating media are more motivated while learning [3].

Instructors find themselves in a dilemma; trying to tie media elements into assignments usually leads to unwanted complexity – especially for introductory courses using the language C++. Students can easily get bogged down in a platform-specific graphics API, losing focus of the original purpose of the assignment – learning arrays.

In the past, different approaches to teaching arrays have been explored. Initial findings indicate that presenting array programming concepts in an alternative format are more popular with students; such formats include using spreadsheets [5], and experimentations with Image Processing [3]. Both of these approaches have the benefit of being inherently visual, and therefore giving immediate feedback. This approach is gaining momentum, and is being implemented in several schools and universities; as we have seen in our classes, it is this immediate feedback which seems to capture student interest as well as give the student a feeling of empowerment. A quote from the paper from Egbert et al on using image processing in the classroom supports our findings.

"Most students did indeed enjoy the visual nature of the project and were surprised that they could write a program to accomplish so much after just one programming course." [2]

We have extended the idea of using image processing to teach arrays to include the area of real-time video. By conceptualizing frames over *time,* it is easy to create filters that rely on previous state to generate a renderable image. It is the concept of history that opens up a unique set of algorithms, including the ability for ghosting and motion detection. Most of these filters can be accomplished in only a few lines of code.

It is interesting to note that static transformations such as blurring, mirroring, and RGB color filters can still be applied real-time, but now take on a new interactive feel. From our experiences, when the first filters were created, the students were reserved in their interaction with the camera; but over time, they were seen aiming their camera either at themselves or their friends, usually grimacing, sticking out their tongue or even pretending to eat the camera. From our point of view, they seemed to be enjoying their creations.

A key component to the success of this assignment *is* the high degree of interactivity; students determine both the object they focus the camera on (usually a human subject such as themselves or a friend) as well as which filters to run via a series of keystrokes. Unlike static image processing, the student is free to examine the effects of the filter *while it is running*, and therefore

have the ability to see the effects on any object in the room. This avoids the tedium of constantly reloading the same image to apply a different filter, and speeds up the process for students who want to capture personalized images and apply a filter to it.

Teaching two-dimensional arrays in this manner gets away from traditional textbook examples such as using simple arrays of primitive data types, and moves into real-world application. Students are captivated by programs that manipulate images, and instantly see that what they are learning has real-life application. This seems to generate excitement, which in turn generates motivation.

2. Implementation

At our institution, we have students with varying programming backgrounds and interests ranging from those who are specializing in software development (a track in our IT degree) to those who are only taking the course because it is required. The Department of Information Technology is home to software development at our institution, and we teach CS1 and CS2 courses in addition to intro and intermediate programming for IT students; additionally, we offer five courses at the senior level for those students who are specializing in software development. Because it was experimental, the first pass at using this technique occurred in Fall 2002 in a course of upper-division software development students with a class size of 12. The same programming assignment was given to introductory programming students in Fall 2003 and Fall 2004, which had class sizes of approximately 25.

The overall lecture time spent on 2D arrays and image manipulation was one week; while this isn't much time, we are not able to dedicate more time given the constraints of other material that must be covered. Simple filters like negative (inverted), red, and grayscale were created live during the first lecture. The algorithms for the more complex filters, such as ghosting, motion detection and sinusoidal distortion were discussed during the following lecture.

The idea of using image processing to teach 2D arrays originated from working with the ARToolkit [6], which was developed for creating augmented reality applications. It provides the skeleton needed for video capture, camera recognition, and key events for callback functions. Additional functionality such as fiducial (marker) recognition and tracking can be discarded – freeing up resources for more complex transformations.

The main benefits to using the ARToolkit as a video capturing system are:

- The frame capturing code is already written
- Key event capturing for callback functions
- Multiple platforms, such as Windows, Linux and SGI – avoiding complex windowing and graphical APIs
- Works with multiple types of cameras (camera recognition)
- Multiple implementations for Windows, based on Microsoft Vision SDK, OpenGL and DirectShow technologies
- A Java version exists [7]

From an instructional point of view, the ARToolkit solves many problems related to an introductory assignment in programming courses, especially for students with different hardware and different operating systems; having Linux and SGI versions available broadens the potential student base. For consistency, we chose to work with the C++ implementation using Visual Studio, and provided help installing the appropriate software and setting up their IDE. USB cameras were used for their high frame rate, with the two most popular being produced by Logitech® and Intel®.

Burger notes that there are "a plethora of file formats" such as JPG, GIF, and BMP, and that "writing image codecs (encoders-decoders) to handle all of these formats would be nearly impossible" [1]. Using video capture avoids many of the problems that stem from this – as file format and compression have not yet been applied; students are working with "raw" pixel arrays.

3. Abstractions

The ARToolkit leaves the student with a clean, one-dimensional data pointer to the frame buffer – essentially an array of raw data. Any changes to values within the data pointer will be immediately updated at the next drawing of the frame. From a pedagogical point of view this is *great*, because it gives us the liberty to cater the assignment to students at different levels by providing an appropriate amount of starting code, if any. It was from this data pointer that we began to build the filters.

Because the image processing assignment is used in two different courses, an appropriate level of abstraction is necessary. We therefore provide varying amounts of beginning code, such as giving functions for accessing and modifying values in a two-dimensional fashion (*putPixel* and *getPixel*). Two copy functions are provided as well - the *copy* function, which duplicates the original frame (*dataPtr*) into a copy (*dataPtrCopy*), and *copyBack*, which performs the opposite.

The difficulty of filters given to students varied as well. Filters such as the red, green, blue, negative, and grayscale are relatively similar and simple to implement. Filters such as mirroring, blurring, randomizing, chromakeying, and ghosting fell into a middle category; the sinusoidal distortion and motion detection into the advanced.

For the advanced group, we felt it important that they become comfortable with more technical array concepts, such as accessing a one-dimensional array given two-dimensional coordinates *i*, *j*, following the general formula:

value = SCREEN_WIDTH**i**4 + *j**4;

Note that the multiplication by four is necessary because the size of one pixel is 32 bits (BGRA).

Filters are then mapped to keys, which when pressed put the application into different modes. If designed correctly, filters can

be applied cumulatively (but only in certain orders). The ability to perform real-time swapping and cumulative application of filters gives the students immediate visual feedback on the correctness of their algorithms and allows them to explore the effects of chaining them together.

4. The Filters

The filters themselves are best categorized by level of difficulty.

For simplicity (disregarding filter chaining), filter code is placed in a series of two nested loops, where the variable *i* goes from 0 to SCREEN_HEIGHT, and *j* goes from 0 to SCREEN_WIDTH. The variable *value* is calculated from the previous formula, and *dataPtr* is the pointer to the frame buffer. In cases where a secondary buffer is needed, a second frame buffer, *dataPtrCopy,* is used. The *dataPtrCopy* variable can then be copied back to be rendered, or in the case of chaining, be passed to another filter for further processing.

4.1 Static Image Processing

From a pedagogical point of view, applying simple filters to video is nothing new. Essentially, the student manipulates the frame buffer in the traditional way by directly altering values within the array. These filters are really used to ease the student into the filtering concepts, and are still enjoyable because of their interactivity. All of these filters essentially work the same way, by changing individual values within the array, and are basically "freebies" for the class. These examples can easily be extended to create other simple filters,

4.1.1 Thresholding, Red, Blue, Green, Negative, Grayscale

As you can see from the example below, creating these filters is simply a matter of writing new values into the original data pointer.

```
if (mode == negative) {
  dataPtr[value]   = 255 - dataPtr[value];
  dataPtr[value+1] = 255 -
                     dataPtr[value+1];
  dataPtr[value+2] = 255 -
                     dataPtr[value+2];
}
if (mode == blue) {
  //dataPtr[value] = dataPtr[value]; // B
  dataPtr[value + 1] = 0; // G
  dataPtr[value + 2] = 0; // R
}
```

4.1.2 Sinusoidal Distortion

The sinusoidal filters distort the image in a sinusoidal fashion, much in the same way "lens" screen savers work. These filters became known to my students as the "Arnie" filters (named after Arnold Schwarzenegger), mainly because of the box-like effects on the human face. While this certainly could be considered a static image manipulation algorithm, the interactivity level of this filter is very high. Three independent filters were created for this, including horizontal stretching, vertical stretching, and a combination of both horizontal and vertical stretching.

Figure 1. The Sinusoidal filter in action

These filters require both a horizontal and vertical lookup table, which represent offsets, or shift values for the new pixel. Note that this table only needs to be calculated one time. Values range in a sinusoidal manner from 0 to -LENS_EFFECT half way through this array, and LENS_EFFECT to 0 through the remainder of the array (where LENS_EFFECT is the maximum shift amount). Intuitively this makes sense: pixels towards the middle should be shifted the nearly the maximum amount away from the center, while pixels on the surrounding edges should moved very little, if any.

```
for (int i=0; i< SCREEN_WIDTH; i++) {
   sinXArray[i] = (int)(sin(
    ((double)i/(double)SCREEN_WIDTH)
    *2*3.1415926)*LENS_EFFECT);
    sinXArray[i]*=4;
}
```

Once the lookup table has been calculated, the effect can be created simply by adding this offset to the original slot value, and copying its contents into the new frame.

```
if (mode == horiz) {
     dataPtrCopy[value] =
       dataPtr[value+(int)sinXArray[j]];
     dataPtrCopy[value+1] =
       dataPtr[value+1+(int)sinXArray[j]];
     dataPtrCopy[value+2] =
       dataPtr[value+2+(int)sinXArray[j]];
}
```

An interesting, yet easy "bonus" feature of the lookup table is that it can be recalculated, having LENS_EFFECT be variable; you can then bind keys such as + and – to amplify the effect, providing "zoom in/out" functionality.

4.1.3 Chromakeying

Chromakeying is the process of replacing all instances of a given color range with some other color, or with some information from another source, such as a bitmap. Most graphic formats usually require the stripping of header information, as well as decoding. For full credit, the students were required to key off the color blue, and replace it with the color red.

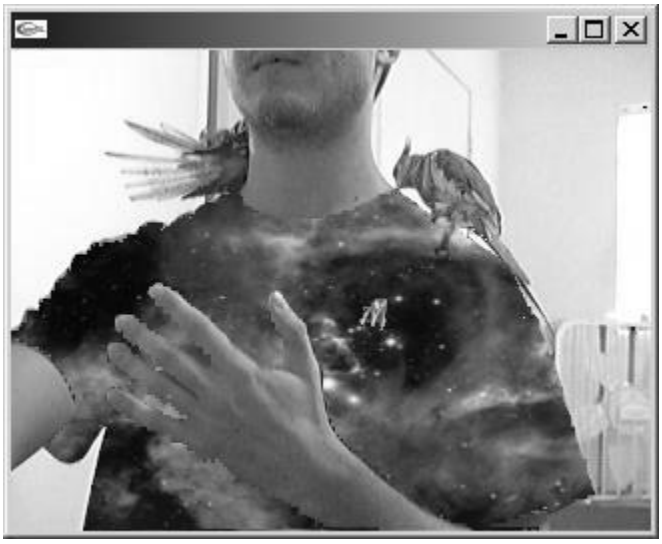

Figure 2. The chromakeying filter

A few students took it upon themselves to chromakey using information from a bitmap instead of the plain color red (Figure 2). Some of the ideas they came up with were weather broadcasting, and creating a "portal".

4.2 Filters Requiring Previous Frames

4.2.1 Ghosting

One of the most popular filters was the *ghosting* filter, which created a dream-like or "smearing" appearance. This filter is created by adding in only a fraction of the current frame into an average, and copying the new average into an additional frame; to achieve this, a secondary buffer needs to be created.

From a visual standpoint, this algorithm appears complex, but can be accomplished in only a few lines of code:

```
if (mode == ghost) {
dataPtrCopy [value]=((GHOST_FACTOR-1) *
   dataPtrCopy[value] +
   dataPtr[value])/GHOST_FACTOR;

dataPtrCopy[value+1]=((GHOST_FACTOR-1) *
   dataPtrCopy[value+1] +
   dataPtr[value+1])/GHOST_FACTOR;

dataPtrCopy[value+2]=((GHOST_FACTOR-1) *
   dataPtrCopy[value+2] +
   dataPtr[value+2])/GHOST_FACTOR;
}
```

Figure 3. The ghosting filter

Having the variable GHOST_FACTOR allows the student to vary the relative "depth" of the ghosting, and can also be bound to keys to change its value.

4.2.2 Motion Detection

The motion detection filter is a great example of how previous frames can be used, and is an extension of the ghosting algorithm. An average is used and compared against the current frame. Current pixels that differ from the average very little are grayed-out, while current pixels that deviate too much are exposed. This tolerance can be adjusted as well, with a tight difference similar to a motion edge-detection algorithm.

Figure 4. The motion detector filter

5. Feedback

The overall feedback received from the students was very positive. Students seemed to enjoy the interaction with the filters, and felt a sense of confidence in their coding skills. Some of the comments included:

"The filters assignment was probably my favorite of the class. Not only was it interesting to see such dramatic results from my coding, but it also boosted my confidence and kept me motivated. It is really fun to be able to interact with your own program that way."

"It was a fun challenge to think of an effect that we wanted to produce, and then develop and optimize the algorithm to achieve the desired effect. Seeing the pixels on the screen was also an interesting way to visualize how the data in memory was being altered as it ran through the filters."

"I thought that coding up the filters was a great way to learn how to manipulate 2D arrays … The concept is simple enough to grasp, but putting it into practice taught me a lot…"

6. Conclusions

Creating real-time video filters is an engaging and effective way to teach programming concepts. Doing things in real-time opens up a unique set of algorithms and allows traditional image processing techniques to become interactive applications. Students were able to perform well on the assignments and seemed more active learners during the presentation of the concept of 2D arrays.

We are leveraging from our experience using 2D real-time filters in an introductory C# programming course this summer (2005) and plan to use it for future programming courses in the fall and spring. The difference in the C# course is that we will be using static-image manipulation that does not require the camera (to cut down on the cost to the student); we hope that students are still engaged and empowered by manipulating images and learning fundamental constructs such as 2D arrays.

7. ACKNOWLEDGMENTS

Special thanks to our students, especially Greg Goss, Brad Dameron and Jeremy Brooks for their help.

8. REFERENCES

[1] Burger, Kevin R., Teaching Two-Dimensional Array Concepts in Java with Image Processing Examples. SIGCSE '03 (Reno NV, Feb. 2003).

[2] Egbert, Bebis, McIntosh, LaTourrette, Mitra. Computer Vision Research as a Teaching Tool in CS1. 32nd ASEE/IEEE Frontier in Education Conference (Boston MA, Nov. 2002)

[3] Guzdial, Mark. A Media Computation Course for Non-Majors . ITiCSE '03 (Thessaloniki Greece, 2003).

[4] Hunt, K. Using Image Processing to Teach CS1 and CS2

[5] Warren, Peter. Teaching Programming Using JavaScript http://www.ics.ltsn.ac.uk/pub/JICC5/warren%20js11.doc

[6] Billinghust, Mark and HITLab. http://www.hitl.washington.edu/research/shared_space

[7] Geiger, Ch., Paelke, V., Reimann, Ch., Stocklein, J. JARToolkit – A Java Binding for ARToolkit. First IEEE Workshop on ARToolkit, ART02, (Darmstadt Germany, 2002)