New Tools to Enable Children to Manipulate Images
Through Computer Programming

by
Evelyn Eastmond

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
May 26, 2006
Copyright 2006 Massachusetts Institute of Technology. All rights reserved.

Author

Department of Electrical Engineering and Computer Science
May 26, 2006

Certified by
Mitchel Resnick
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses
New Tools to Enable Children to Manipulate Images

Through Computer Programming

by

Evelyn Eastmond

Submitted to the
Department of Electrical Engineering and Computer Science

May 26, 2006

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis proposes a programmable toolkit for building image filters to make the concepts of basic image processing accessible and engaging to children. The toolkit was developed in the context of Scratch, a media-centric programming environment intended for children. For this thesis, I redesigned the image manipulation framework in Scratch to ‘open the black box’ of image processing. Through user testing done with children, it was shown that the new framework sufficiently exposed the image-processing concepts so that children were able to create their own compelling image-processing algorithms. Although this framework was not ideal for the Scratch programming environment, the concepts explored in this work can be used in classrooms that teach introductory computer programming through media computation, or in a separate programming environment for novices which focuses on basic image processing.

Thesis Supervisor: Mitchel Resnick
Title: Director of the Lifelong Kindergarten group at the MIT Media Laboratory
ACKNOWLEDGMENTS

I would like to thank the following people for supporting me throughout my time in the Lifelong Kindergarten group, from my UROP years all the way through my Master of Engineering thesis.

- Anne Hunter and the MIT EECS department for handling all logistical and administrative details of my course requirements, and for kindly pointing the way whenever I felt lost.

- Members of the Lifelong Kindergarten group at the MIT Media Lab for graciously accepting me into the group and making me feel like part of something meaningful.

- My mother, John, Lawrie, family and friends for continuously supporting me and believing in me when I most needed it.

- John Maloney for teaching me everything from Smalltalk to how to be a good programmer, student, communicator, team member and everything in between. Thank you for setting a great example and for being so patient with me.

- Mitchel Resnick for allowing me to do what I love, caring about me as a person, and giving me the opportunity to work on Scratch, the most gratifying project I've ever worked on. Thank you so much for letting me become part of your group.
# CONTENTS

I. INTRODUCTION ................................................................. 8

II. CURRENT IMAGE MANIPULATION IN SCRATCH .................. 14

III. OPENING THE BLACK BOX: ADDING IMAGE PROCESSING TO SCRATCH 18

IV. IMPLEMENTATION ......................................................... 24

V. USER STUDY ................................................................. 34

VI. CONCLUSION .............................................................. 44

VII. FUTURE WORK ........................................................... 52

REFERENCES .................................................................. 55

APPENDIX A: USER STUDY PRESENTATION
I. INTRODUCTION

The Best of Both Worlds

Growing up, I always viewed computers as tools with which to express myself artistically. Whenever I came in contact with a computer, I immediately searched for its primitive paint application. I would sit for hours and marvel at how quickly and easily I could sketch out my ideas, erase them, and start over again. As I got older and computers got more sophisticated, I was exposed to more complicated artistic software. These new programs offered me more tools and knobs with which to make my artistic creations even more interesting. I often found myself playing with the image filtering features because I enjoyed how they dramatically changed my digital images with just the press of a button. I would take an image and filter it over and over to see what exciting effect would come out next.

However, after I had played with these filter effects for long enough, I started wondering how they actually worked. What information from an image allowed the creators of the software to change my image in different ways? If I had access to that same information, what kinds of effects could I come up with? I felt that if I had access to that information, then I could control the filters and the images in the new ways that I envisioned. The predefined filters felt limiting to me and I felt compelled to make them my own. I eventually became a programmer because I enjoyed learning about technology, though I never lost my motivation to create art on the computer.

In this thesis work, I present a programmable toolkit for building image filters to make the concepts of basic image processing accessible and engaging to children. The work is carried out in the context of Scratch,[1][2] a media-centric programming environment developed by the Lifelong Kindergarten group at the MIT Media Lab. By adding a basic image-processing framework to Scratch, my goal is to combine the worlds of digital artistic expression and programming at the hands of children.

In sections I and II of this document, I discuss the motivation to add a basic image-processing toolkit to the Scratch programming environment. In sections III and IV, I discuss the design and implementation of the toolkit. In section V, I present the results of using the toolkit with children in a user study. Finally, in sections VI and VII, I present my conclusions and suggestions for future work.
*Children and Programming*

The Lifelong Kindergarten group at the MIT Media Laboratory, led by Mitchel Resnick, specializes in creative methods of learning through technology. The group's work focuses on exploration and ‘tinkering’ as a learning process where children naturally lead themselves to discovery by learning at their own pace and doing engaging, hands-on, technological activities. The overarching goals are “to help kids become more fluent and expressive with new technologies… to help them explore important concepts (often in the domains of mathematics, science, and engineering) through their expressive activities; and, most broadly, to help them become better learners.”[3]

One of the domains in engineering education that the group has focused on is computer programming. The group’s work in developing programming languages for children draws upon previous work by Seymour Papert (LOGO) [5] and Alan Kay (Squeak Etoys) [6] where programming was viewed as a tool for kids to model their own life experiences and learn how to reason about those models in the process. Seymour Papert argues that “when a child learns to program, the process of learning … becomes more active and self-directed. In particular, the knowledge is acquired for a recognizable personal purpose … (and) the new knowledge is a source of power.” [5] Also, programming is viewed as an important educational activity because with it children “learn to think about thinking” [5] and they can apply those same problem solving skills in other educational domains.

There are also important mathematical and technical concepts used in programming that can enhance a child’s education whether they are going to become a programmer or not. Through programming, children learn how to use procedures, rule based systems, symbolic logic, and even coordinate systems. In addition, as computers become an integral part of society and of children’s lives everywhere, it is important to give children the notion that computers are tools that humans can program to do creative tasks as opposed to machines that just feed information. It is important for children to understand that having a computer is like having books, crayons, rulers, pencils, scissors, paper and paint all in one.
Another important element of a child's educational process is creativity. Children learn and become engaged in activities by being artistic and expressing themselves. The Lifelong Kindergarten group has set up a global network of Computer Clubhouses where technologies for self-expression are made accessible to children and teens in a voluntary atmosphere. At these Computer Clubhouses, children learn how to convey their ideas creatively with new technologies by making their own computer animations, robotic constructions, digital photo manipulations and other artistic creations. In this setting, children learn how to communicate by recognizing concepts and ideas that are important to them and finding ways to use the technologies available as communication tools.

In these Computer Clubhouses, one popular activity for children is to use photo-editing software to manipulate their favorite digital images and to create expressive artwork. Software such as Adobe Photoshop, KidPix, Macromedia Fireworks, Bryce, and Maya allow children to be 2D and 3D artists without much training or expertise. Also, in this digital era, it is very easy for children to find source images for their artwork, either by downloading images of famous movie stars or athletes from the Internet or simply by downloading pictures of themselves taken with digital cameras. Aside from using photo-editing software to extract images to make artistic or humorous collages, these children also make use of the image-effects features available in many of these software packages. As I used to do when I was young, children are now constantly using computers to image-process their pictures and turn them into works of art.

Between photo-editing software, the growing use of digital special effects in popular movies and television shows, and the popularity of digital cameras, digital image processing is quickly becoming an everyday artistic form exposed to and used by children. Much like the traditional crayons and markers, computers are becoming important, foundational artistic tools.
Teaching Image Processing through Scratch

To combine the ideas of learning through programming and learning through digital self-expression, the Lifelong Kindergarten group has created a new programming environment for use in the Computer Clubhouses called Scratch. Scratch is a media-oriented programming language for children and teens where there is no text language and instead children stack together building-block commands to create their programs. Scratch also includes extensive media support, which makes it an ideal environment where children learn how to program engaging projects while expressing themselves creatively. Figure 1 shows an example of the Scratch interface, with an example Scratch project and its stacks of building-block commands in the scripts area.

Figure 1. Screenshot of the Scratch programming environment.
One of the many expressive tools in Scratch's set of programming blocks is an image-effect block that children can use to apply various graphical effects to the objects in their projects. This is Scratch’s equivalent to the image-processing features that children are used to using in popular photo-editing software. This programming block gives children the choice of which specific image filter they apply to their objects (fisheye, whirl, etc). The block also takes an input parameter which controls the amount that the effect gets applied. The block can be used just like any other command block in Scratch, therefore it provides a way to programmatically control the image effect. This makes the image-effect block much more appealing than the traditional use of image-filter features in photo-editing software because it adds an engaging dynamic element.

While this image-manipulation block gives children dynamic control over their images, it still does not fully expose what information is being changed and how the image effect is actually working on the data in the image. In this project, it is my goal to give children the power that programming provides by opening the black box of this image-filter block and giving them direct control over the red, green and blue values of the pixels in the image. Scratch is an ideal context in which to carry out my project because it is a child-friendly programming language with image manipulation support. By redesigning the image-manipulation block and exposing some of the underlying concepts of image processing, I hope to give children the power to imagine their very own image effects that perhaps software engineers might never have thought of.
The idea of teaching basic image processing concepts to novices is not new. At Georgia Tech, Mark Guzdial has created an introductory programming course taught entirely through media computation.[4] Mark had noticed that those not majoring in Computer Science were not performing well in his introductory programming courses because the concepts taught were too abstract and “not related to everyday life”, an important connection if students are to gain transferable knowledge.[4] His new media computation approach focuses on programming media in familiar contexts that his students find motivating, such as image manipulation, sound manipulation and Web page creation.[4] Even though his curriculum teaches programming that is exclusively for media computation, it still touches upon foundational programming constructs such as assignments, sequential operations, iteration and conditionals. As a consequence of this work, Guzdial has seen an increase in academic performance by novices in his introductory programming courses.

I use Guzdial’s work as inspiration for my thesis because he recognized the importance of media as compelling content when trying to teach computer programming to novices. My work aims at a similar goal. Ideally, children can learn basic image-processing constructs, the specifics of how digital images are composed on computers, and the power of manipulating this media at a lower level so that they have the power to imagine artistic creations beyond those available in programs like Adobe Photoshop and in Scratch itself.
II. CURRENT IMAGE MANIPULATION IN SCRATCH

A Single Parameter Approach

The command blocks in the Scratch language are split up into categories corresponding to the type of action they perform. Within the ‘Looks’ category, there are blocks that allow children to program different looks (or costumes) for their objects (or sprites). Two of these blocks apply image filters to costumes (refer to Figure 2). One block applies the image effect according to an absolute parameter and the other changes the application of the image effect by a delta value.

<table>
<thead>
<tr>
<th>set</th>
<th>effect to</th>
<th>change</th>
<th>effect by</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="set color effect to 0" /></td>
<td><img src="image" alt="change color effect by 0" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. The current image manipulation blocks in Scratch.

The effect field has a drop down menu with various image effects to choose from: color, fisheye, whirl, pixelate, mosaic, brightness, and ghost (or alpha). To use the set effect to block, you first choose an effect and then enter a value between 0 and 100 (or –100 to 100 in the case of whirl) into the argument field. This number acts as the parameter for the image effect when executing the block. Table 1 lists what the parameter means for every effect.

<table>
<thead>
<tr>
<th>Effect</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>degree along the hue circle that the colors should shift to</td>
</tr>
<tr>
<td>fisheye</td>
<td>degree of the lens refraction</td>
</tr>
<tr>
<td>whirl</td>
<td>amount of whirl and direction (&gt; 0 = right) (&lt; 0 = left)</td>
</tr>
<tr>
<td>pixelate</td>
<td>scale</td>
</tr>
<tr>
<td>mosaic</td>
<td>number of tiles to break up to</td>
</tr>
<tr>
<td>brightness</td>
<td>relative brightness shift</td>
</tr>
<tr>
<td>ghost</td>
<td>alpha value</td>
</tr>
</tbody>
</table>

Table 1. List of image effects and parameters in the current Scratch image manipulation blocks.

These blocks can also accept other mathematical Scratch blocks into the argument field in addition to typed in numerical values. This means that the parameter can take on interactive and changing values while the block is being executed (when combined with loop controls and variables). For some of the
effects, if during execution the parameter value surpasses 100, then the value wraps around to 0 automatically.

Success of the Current Approach

This approach of using a single parameter to control the image effects in Scratch has been successful. One reason for this is that the blocks work in the context of all the rest of the Scratch command blocks, so they are easily discoverable and usable. A child can put these blocks anywhere in their program and there is usually an immediate, visible change to the object’s costume. Because of this smooth integration into Scratch and the fun of discovery that the blocks afford, they have been an integral part of the Scratch language.

Another reason for the success is that playing with a single parameter is simple and yet powerful enough to give varying effects. A child is able to control a parameter of 'how much' the effect gets applied. It takes playing and tinkering with the blocks to figure out what 'how much' really means and what it is affecting. Once the child figures out what the parameter means, then he/she can programmatically control that value with variables, math blocks, loops or sensor input blocks available in Scratch (such as mouse x and y coordinate reporters). Therefore, programming with the blocks is engaging yet educational for children especially since it is so easy to get started and the results are immediately visible.

After looking through various Scratch projects made by children and looking for trends in the way that they use the image manipulation blocks, I found that the dynamic potential of the single parameter value in the argument is often used creatively. For example, the most commonly used image-effect block is change color by, usually in the context of a forever loop. This effect is primarily used on bright, multi-colored clip art. By continuously applying the change color by block to one of these colorful graphics, the image changes a great deal because all of its colors keep changing hue by some amount. This is one of the ways that children use the single parameter to create interesting and interactive effects. Figure 3 shows an example of that block being used and the effect that it created with a colorful image from the Scratch image library.
These blocks are successful because the single parameter is so easy to change and yet provides dynamic and beautiful results. Also, by tinkering with these blocks, children learn how different parameters have different effects on their images.

*What Is Hidden by the Single Parameter*

While the current image-effect blocks in Scratch are educational and exciting, they hide away the important details and lower level concepts of what it means to change an image on a computer. Although there is a parameter that children can tweak to get the effect that they want, the value means something different for each effect. This might get confusing for children as they try to explore all of the effects at once. While using negative numbers with the whirl effect causes the whirling to happen in the left direction versus the right direction, using negative numbers with the pixelate effect is equivalent to using the corresponding positive number. What the value actually affects, and *how* it affects it, is hidden by the simplicity of the block. It is not clear that some logic is going on behind the scenes that take absolute values of the inputs sometimes and sometimes not. It is not clear whether the parameter means the degree of the hue on the color wheel, the refraction angle of light through a lens, or some other metric.

Simply put, the current image-manipulation blocks in Scratch are black boxes as to what is going on inside image filters. Mitchel Resnick and Brian Silverman, developers of countless children’s software and engineering construction kits, state that “the choice of the basic building blocks in a programming
language determines what kids are likely to learn as they use the language.” [3] When designing artistic blocks within these programming languages, they felt the best way to deal with the complication of three color dimensions would be to get rid of two of them and just have one parameter that ranges from 0-100. [3] They then go on to say that “if [their] primary goal were to help kids learn about red-green-blue composition of light, then [a] single-input setcolor command would be a bad choice.” Even though I agree with them that “the single-input setcolor encourages and supports much greater exploration of color effects than the three-input version,” for this project, it is the goal to teach children about the composition of light and the composition of digital images, so the single-input approach is not well suited.

If children have been learning about color theory for years in school, why not, as we move into the digital age, teach them about color for digital purposes? It is important to start answering questions for children such as: What are the elements that make up a digital image? How are those elements changed in order to change the image as a whole? How does one change an image’s shape, color and size? These are the questions that the current blocks in Scratch do not answer, yet their answers could teach children how to use computers and programming to create new types of digital art.
III. OPENING THE BLACK BOX: ADDING IMAGE PROCESSING TO SCRATCH

Design Principles

Along with the idea of opening up the black box to expose image-processing concepts, there were two other guiding design principles that I had in mind while changing the image manipulation framework of the Scratch programming language. All three design principles come from the lessons learned from the current image-manipulation blocks and my motivation to give children powerful digital image-manipulation tools.

These design principles are:

– Open up the black box but keep unnecessary details hidden
– Fit naturally into Scratch
– Keep it interesting and engaging

I have already discussed why the first point is important. The second point is important because of the language of the system. I wanted to use Scratch as an avenue for this thesis work and it was important to stick to a familiar language and programming metaphor that children were used to. Finally, as Guzdial's work [4] shows, the act of programming isn't compelling for novices unless there is motivation behind the thought process. If children find image filters interesting and engaging, then they will want to learn how to think through the image processing to get what they want.

Open Up the Black Box but Keep Details Hidden

The main goal when redesigning the current image-manipulation blocks in Scratch was to open up the black boxes and expose the core image-processing concepts that are involved in simple digital image manipulation. These basic concepts will start teaching children the important notion of how to change a digital image and what the necessary steps are. However, it was important to choose which black boxes to open, so potentially confusing and not necessarily educational concepts were kept hidden.
The Concepts to Expose

The specific image-processing concepts that children learn with these blocks are much like the ones that Guzdial teaches in his book.[4] First, children learn about the digital makeup of images as pixels, or picture elements. Children come across this concept in their everyday lives because of digital photography, photo-editing software and computer games, so it is an important fact for them to learn. Next, they learn about the color makeup of pixels on a screen. Color mixing and color theory are concepts that are taught to children early on in education. In art class, children are taught that the primary (subtractive) mixing colors are blue (cyan), red (magenta) and yellow. Now, as we move into an age where children are learning to manipulate digitized media, it is important to teach them that colors on screens behave much differently from what they have learned. By creating a framework in which children manipulate the red, green, and blue levels of pixels, they can start becoming comfortable with this concept. Finally, children will learn the basic logic of image processing by learning that a repeated process is applied to all the pixels of an image and that this creates an overall effect. This simple concept is powerful and gives children the capability to think in small steps.

As an experiment, another image processing concept that I chose to expose was the relative position of each pixel in the 2D image space, stored in x and y coordinate values. This spatial information is useful for creating generative effects and gives insights into the placement of pixels in relation to each other.

The Concepts to Keep Hidden

There are some lower level image-processing concepts that were kept hidden because they did not add educational value to the concepts learned and could potentially be confusing. One of these concepts is the raster-scan method for accessing the pixels in an image. Traditionally, it is taught that you access the pixels in an image with for loops, control constructs that repeat a process a specified number of times (i.e. for each pixel). While embedded for loops are important for children to use when learning programming, they are not necessary to grasp the concepts of image processing. For the purposes of the image filters, it was easier to abstract this concept away and make a general for all pixels loop structure which would apply its embedded stack of blocks to all the pixels in an image at once.
Another concept which was abstracted away was the idea that each pixel's red, green, and blue color channels can hold up to 256 values because of the specifics of computer storage. This concept is not tied to image processing aside from the structure of computers, but what is important is that the color values lie in a range. A more natural range that children are used to dealing with, 0-100, was chosen for this framework. This makes it easier for children to understand what values correspond to 'no red' (0), 'half red' (50) and 'all red' (100).

*The Concepts Not Included*

Another aspect of image processing that is appealing is the use of the spatial information of pixels to change the geometry and shape of an image. Also interesting are convolutions and using pixel neighborhood information to create effects based on relationships among colors. These concepts were not explored in this work because it was important to first explore the feasibility of exposing simple color theory and basic image manipulation. The geometry and convolution aspects of image processing would be the next step that a child could learn when going through this framework. By simply learning how digital images are composed and changed while also learning a new color theory along the way, children are already given enough new tools with which to create art on a computer.

*Fit Naturally Into Scratch*

If these blocks are to become part of children’s every day language, then it is valuable to put the system in a context and a language that the children already understand. The Scratch project seemed like the perfect language because it offers a text-free, novice-oriented programming environment where a child can learn core computer science concepts and be expressive at the same time. Ideally, the blocks in this new framework would have a similar structure, interaction and programming metaphor so that they become just another creative tool within Scratch.

The Scratch programming model is a visual, command-block language where each block executes an action on objects (sprites) in a 2D virtual world (stage). These commands can take parameters to control how they behave. Commands are then stacked together to construct programs which execute linearly in the order in which they are connected. To simulate loops, there are special C-shaped blocks
which contain a stack of blocks to be executed some repeated number of times. The language also contains conditional structures which execute embedded code only when special Boolean blocks evaluate to true. To connect Scratch programs to a launch button, and for inter-object communication, there are special ‘hat’ blocks which sit on top of scripts to act as labels.

As arguments to the command blocks, the Scratch language accepts typed in numbers or special blocks called reporter blocks that return numerical values. Some reporter blocks are variables which the child can create and increment. Other reporters retrieve special values which the child cannot control, for example the mouse x and y coordinate position or sensor inputs. There are also mathematical blocks which can act on these number values, and those mathematical blocks can themselves be placed inside argument fields to report the output values. Figure 4 shows examples of typical Scratch programming constructs.

Figure 4. Typical Scratch programming constructs including control structures: forever and repeat loops, command blocks, hat blocks, mathematical functions and numerical value reporter blocks.

While redesigning the image-manipulation blocks in Scratch, it was important to design them similarly to the blocks shown above in order for them to fit naturally into the Scratch programming metaphor.

**Keep It Interesting and Fun**

The final guiding design principle was that the content and use of the blocks be fun and compelling for the children. This is important because, unless the children have motivation or a final goal they want to achieve, then they will not be interested in the process of getting there.
Within the limited scope of image-processing concepts explored for this project, it was unclear whether it would be possible to generate enough interesting effects for the children to engage in. Since only color information can be affected, they would be missing out on some of the more complicated image effects that were already possible in the Scratch language: whirl, pixelate, mosaic and fisheye. At the same time, it was shown earlier that children already make great use of just the color changing effect block because of the interesting and surprising effects that can result when used with the appropriate graphics. The hope is that by being given a limited set of tools to create countless color effects, children can come up with the exciting image filters themselves. The following are some examples of the image filters that I envisioned would be possible with this framework.

**Color Manipulation with Single Numeric Values**

By setting colors to typed-in numerical values, children can directly explore the red-green-blue color space. Examples of possible effects include: remove red, increase green, remove red and green (colorize to blue), or set all colors to 100 (white) for invisibility against the stage. If children want to get more creative, they can use red, green, and blue reporters which contain a pixel’s original information for effects which play off of existing values.

**Color Manipulation with Math and Logic**

By using numerical and logical operators, other color combinations become possible. For example, children can create a gray-scaling algorithm which takes the average of all the pixel components. They can also use the random generator block in Scratch to get a surprise color for every pixel and generate glittering effects. They can also generate a ‘negative-photo’ effect by simply subtracting the color values from 100 to get the complementary colors.

**Color Manipulation with Conditionals**

With conditionals, children can start creating more advanced image effects, like applying sepia tones and posterization. Both of those processes use different threshold levels to get colors to behave a
certain way. The conditionals can also be used to test boundary limits on the x and y position of the pixels, creating image filters which only affect part of an image.

Interactive Color Manipulation & Using Other Scratch Programming Blocks

Finally, by using the reporter blocks already present in Scratch, such as `mouse x` and `mouse y`, children can experiment with dynamic values to control their colors and get interesting effects that change interactively. Scratch also has variable reporter blocks that can change over time and can be used to create dynamic effects.

The Final Set of Blocks

With all of these design principles in mind, I designed a set of image-processing blocks for use in the Scratch language. This framework gives children the ability to: loop through all the pixels of an image, access the red, green, blue, x and y values of each pixel, and then reset the red, green and blue values of to change the pixels and cause image effects. Figure 3 shows the final set of blocks.

![Figure 3. The blocks in the new image-processing framework in Scratch.](image)

For the loop block, a standard C-block was placed within the Scratch ‘Looks’ category that abstracted away the 2D raster scan method of accessing the array of pixels into one convenient block. When evaluated, the script inside this C-block is evaluated on every pixel of the current costume of the selected sprite. The red, green, blue, x and y accessor blocks were placed under the ‘Sensing’ category. Finally, for actually setting the red, green and blue values of a pixel, command blocks that take numerical arguments were placed in the ‘Looks’ category. A separate, all-in-one red-green-blue setter block was created because for some filters, such as gray-scale, it is useful to have.
IV. IMPLEMENTATION

Implementation Overview

Implementing the image-processing blocks in the Scratch language was a simple process because most of the scaffolding was already in place and the task was just to extend the set of existing Scratch blocks. For image-processing algorithms, performance (speed) is always an issue because of the substantial number of pixels in most images. Scratch is written in Smalltalk, which is an interpreted language and therefore not ideal for executing commands repeatedly for all the pixels in an image. Most computationally expensive procedures are implemented in C++ and then loaded into Scratch through a dynamically linked library.

For the purposes of this project, I first implemented the new blocks and their execution framework in Smalltalk to analyze their feasibility and usability. Then, once those blocks were evaluated and finalized, I ported the code to C++ and wrote an opcode translator to make execution of the blocks fast enough for actual use. One motivation for this performance increase is the use of dynamic image filters (such as ones that change with mouse input) which need to be executed quickly enough for smooth interaction. Another motivation is that some costumes in Scratch, such as the stage costumes, can be as big as 480x360 pixels.

Below I discuss the implementation details for the first round of blocks in the Smalltalk language and then I discuss the performance issues and how I addressed them with a second round of implementation by including a primitive C++ plug-in. At the end of this section I present some of the varied image filters that I was able to create after implementing the framework.

Scratch Blocks Round 1: Scratch Interpreter

To incorporate the image processing blocks into the Scratch language, I had to first modify a method in the Scratch object initialization that creates all the blocks and categories. The first thing I had to do was add the block descriptions for **for all pixels**, **set RED**, **set GREEN**, **set BLUE** and **set RGB** under the ‘Looks’ category, and then the **red**, **green**, **blue**, **x**, and **y** getters under the ‘Sensing’ category.
After including the block skeletons in the set of commands available to sprites and the stage, I had to put in the plumbing for interpreting these blocks in the Scratch process evaluator. Because the **for all pixels** C-block needed to interpret the blocks inside of it and then apply it to every pixel in the current costume, that block had to be evaluated differently than most blocks in Scratch. For this, I needed to add a particular evaluation case to the regular Scratch evaluator. When the regular Scratch evaluator came across one of the **for all pixels** blocks, a customized function would take over evaluation. First, it would take the current costume of the sprite and make that the input data for the image filter. Then, it would loop over every pixel and interpret and evaluate every block inside the **for all pixels** loop on that pixel. The pseudo-code for the overall execution is in Figure 4.

```plaintext
if block is a 'for all pixels' block
    then for all of the pixels in the current costume
        for all blocks inside the 'for all pixels' block
            interpret the block
            evaluate it on the current pixel
```

Figure 4. Psuedo-code for the **for all pixel** block evaluation.

To evaluate the individual image processing blocks, I again needed to write a customized function. For this interpreter, I tried to model the existing Scratch evaluator as closely as possible, as well as directly rely on it when appropriate for efficiency. First, I had to check if the block was of a certain kind (**set RED**, **set GREEN**, **set BLUE**, **set RGB**, **red**, **green**, **blue**, **x**, **y**). If it was, then I evaluated it directly on the pixels. If it was any other block that Scratch already knew how to handle, such as a numerical argument or a reporter block, then I passed it on to the Scratch evaluator and used the result. For simplicity, I did not support conditionals in this first round of implementation.

This method of first interpreting and then evaluating the image-filter blocks for every single pixel caused significant performance lags. Table 2 shows the evaluation time for image-processing scripts of different lengths run on images of different resolutions. The overall Scratch stage is 480x360 pixels, which is the largest image possible in the Scratch environment. The smallest useful size for a sprite is that of the default Scratch cat, which is 73x36 pixels. The values were recorded on a Windows XP
machine with 2.19 GHz processor and 896 MB of RAM, which is a typical set up for the computers that children use at the Computer Clubhouses.

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Number of Blocks</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>480x360</td>
<td>1</td>
<td>4734 ms</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>29657 ms</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>82614 ms</td>
</tr>
<tr>
<td>260x345</td>
<td>1</td>
<td>2388 ms</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>16300 ms</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>40472 ms</td>
</tr>
<tr>
<td>73x36</td>
<td>1</td>
<td>39 ms</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>258 ms</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>635 ms</td>
</tr>
</tbody>
</table>

Table 2. Performance timings for evaluation of the blocks using the Smalltalk interpreter.

As the table shows, the evaluation time for a script with 21 blocks on a stage costume (480x360) is about 82 seconds. For an interactive image filter, this would result in a frame rate of (1/82)fps. This performance is not acceptable for simulating interactive image filters. The lag is extremely noticeable and undesirable.

This first round of implementation was successful in that it proved that the blocks were feasible in Scratch as I was able to create image-processing scripts that applied interesting image effects to costumes. Even though I did not have the conditional blocks implemented and the performance was slower than ideal, it was still useful as a feasibility test.

However, before testing the blocks with children and evaluating their usage and learnability, it was essential to make the blocks faster and more responsive. To increase the speed of evaluating the blocks, the interpretation step at every pixel had to be optimized.
Scratch Blocks Round 2: Primitive Plug-in

The repeated interpretation step in the first round of implementation was slow and cumbersome. There was no need to translate the same set of blocks for every pixel. Instead, it would be faster to interpret just once, before running through the entire pixel loop, and then use the interpreted sequence to evaluate the blocks on every pixel.

To accomplish this, I needed to create customized opcodes for translation of sequences to be passed on from Scratch to a primitive plug-in written in C++. The first step was to define the opcodes and instruction set with the necessary functions. I knew functionality was needed to set and retrieve the red, green, blue, x and y values of every pixel. Also needed were all the basic numerical and logical operations in order to perform mathematical manipulations on the pixel values. Additionally, to give flexibility to the types of filters possible, support for a conditional statement was added. Table 3 shows the final instruction set.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Noop</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>/</td>
</tr>
<tr>
<td>5</td>
<td>random(a,b)</td>
</tr>
<tr>
<td>6</td>
<td>%</td>
</tr>
<tr>
<td>7</td>
<td>if</td>
</tr>
<tr>
<td>8</td>
<td>&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;</td>
</tr>
<tr>
<td>10</td>
<td>=</td>
</tr>
<tr>
<td>11</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>!</td>
</tr>
<tr>
<td>14</td>
<td>set variable</td>
</tr>
<tr>
<td>15</td>
<td>change variable</td>
</tr>
<tr>
<td>Opcode</td>
<td>Instruction</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>16</td>
<td>get red</td>
</tr>
<tr>
<td>17</td>
<td>get green</td>
</tr>
<tr>
<td>18</td>
<td>get blue</td>
</tr>
<tr>
<td>19</td>
<td>get x</td>
</tr>
<tr>
<td>20</td>
<td>get y</td>
</tr>
<tr>
<td>21</td>
<td>set red, green, blue</td>
</tr>
<tr>
<td>22</td>
<td>set red</td>
</tr>
<tr>
<td>23</td>
<td>set green</td>
</tr>
<tr>
<td>24</td>
<td>set blue</td>
</tr>
<tr>
<td>25</td>
<td>max(a,b)</td>
</tr>
<tr>
<td>26</td>
<td>min(a,b)</td>
</tr>
</tbody>
</table>

Table 3. Customized opcodes and instruction list used to evaluate the blocks.

A variable array was also needed to store all the ‘registers’ of information needed for this instruction set. In the Smalltalk code, a function interpreted the block list to be evaluated into an instruction list while at the same time keeping track of registers which hold values to be used throughout evaluation. For simplicity, the first three values of the registers were reserved for original red, green, and blue values and the next three were reserved for the final output channels of the pixel. The opcode sequence and the variable array were stored as an integer array and a float array, respectively, for compatibility with the C++ interpreter plug-in.

The only special case during evaluation was the conditional if statement. Translating a conditional statement included the extra step of calculating the jump offset on conditional branching, and inserting that number into a register. One possible optimization that could have been included was constant folding. Constant folding means that the Boolean condition of the if statement is evaluated in Squeak before being interpreted into an opcode sequence. This preemptive evaluation allows one to simply include or not include the instructions in the conditional statement depending on the result. This could mean that the interpreter plug-in would never need to evaluate the conditional statement. However, due to lack of time, constant folding was not implemented.
Essentially, this implementation is a customized register machine for optimized pixel loop evaluation. After generating the opcode sequence and register array for the set of blocks that defined an image filter, Smalltalk could pass the opcode sequence, registers, and the costume bitmap into the C++ plug-in. In the C++ plug-in, there was a switch statement that performed the appropriate actions with the values, registers and pixels at every instruction. As a result of this optimization, the performance greatly increased. The new performance values are shown in Table 4.

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Number of Blocks</th>
<th>Round 1 Times</th>
<th>Round 2 Times</th>
<th>Performance Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>480x360</td>
<td>1</td>
<td>4734 ms</td>
<td>15 ms</td>
<td>315.6x</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>29657 ms</td>
<td>89 ms</td>
<td>333.2x</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>82614 ms</td>
<td>226 ms</td>
<td>365.5x</td>
</tr>
<tr>
<td>260x345</td>
<td>1</td>
<td>2388 ms</td>
<td>5 ms</td>
<td>477.6x</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>16300 ms</td>
<td>39 ms</td>
<td>417.9x</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>40472 ms</td>
<td>98 ms</td>
<td>413.0x</td>
</tr>
<tr>
<td>73x36</td>
<td>1</td>
<td>39 ms</td>
<td>&lt;1 ms</td>
<td>78.0x</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>258 ms</td>
<td>1 ms</td>
<td>258.0x</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>635 ms</td>
<td>3 ms</td>
<td>211.7x</td>
</tr>
</tbody>
</table>

Table 4. Performance timings for evaluation of the blocks using the C++ plug-in.

As the table shows, the performance speed was greatly improved from the first round of implementation. On average, the scripts ran 318.94 times faster with this round of implementation. This increase in speed finally made the blocks usable and ready to be tested with children.

**Scratch Blocks Initial Usage**

Once the performance became usable, I was able to test out some of the image effects that I had envisioned would be possible. Before directly testing the blocks with children, I created some example scripts to get a sense of the range of filters that would be possible and how long the scripts needed to be. Below are tables which list example image filters, separated by categories. These example effects give a sense of the range of artistic and educational potential of these blocks.
## Color Manipulation with Single Numeric Values

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set every pixel color to black.</td>
<td>![Code Block](for all pixels set RED to 0, set GREEN to 0, set BLUE to 0)</td>
<td>![Image]</td>
</tr>
<tr>
<td>Remove the red component from every pixel.</td>
<td>![Code Block](for all pixels set RED to 0)</td>
<td>![Image]</td>
</tr>
<tr>
<td>Switch color channels: new red gets old green new green gets old blue new blue gets old red.</td>
<td>![Code Block](for all pixels set RED to green, set GREEN to blue, set BLUE to red)</td>
<td>![Image]</td>
</tr>
<tr>
<td>Generate a gradient by setting the green value to the y position of the pixel and removing the red and blue component completely.</td>
<td>![Code Block](for all pixels set RED to 0, set GREEN to y, set BLUE to 0)</td>
<td>![Image]</td>
</tr>
</tbody>
</table>
## Color Manipulation with Math and Logic

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn into a gray-scaled image using NTSC luminance values.</td>
<td>for all pixels&lt;br&gt;set RED to ( \text{red} + 0.298 \times \text{green} + 0.587 \times \text{blue} + 0.114 )</td>
<td>![Effect Image]</td>
</tr>
<tr>
<td>Generate random noise by setting all colors to a random value.</td>
<td>for all pixels&lt;br&gt;set RED to pick random 0 to 255&lt;br&gt;set GREEN to pick random 0 to 255&lt;br&gt;set BLUE to pick random 0 to 255</td>
<td>![Effect Image]</td>
</tr>
<tr>
<td>Use max and min to choose among various channels when setting the colors.</td>
<td>for all pixels&lt;br&gt;set RED to ( \text{red max green} )&lt;br&gt;set GREEN to ( \text{blue min green} )&lt;br&gt;set BLUE to ( \text{blue max red} )</td>
<td>![Effect Image]</td>
</tr>
<tr>
<td>Subtract every color from 100 to get the complementary colors.</td>
<td>for all pixels&lt;br&gt;set RED to 255 - ( \text{red} )&lt;br&gt;set GREEN to 255 - ( \text{green} )&lt;br&gt;set BLUE to 255 - ( \text{blue} )</td>
<td>![Effect Image]</td>
</tr>
</tbody>
</table>
# Color Manipulation with Conditionals

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a sepia toned image by using a threshold on the amount of red in every pixel.</td>
<td>![Blocks for Sepia Toned Image]</td>
<td>![Effect Image]</td>
</tr>
</tbody>
</table>

For all pixels:
- set RGB to \[\text{red} + 0.255 + \text{green} + 0.255 + \text{blue} + 0.16 \] | <br>**if** red < 13 <br>set RED to red + 13 <br>set BLUE to blue + 0.55 <br>**if** red > 63 and red < 192 <br>set RED to red + 1.53 <br>set BLUE to blue + 0.65 <br>**if** red > 191 <br>set RED to red + 103 <br>set BLUE to blue + 0.95 |
| Create a black and white posterized image by using a threshold on the brightness of every pixel. | ![Blocks for Black and White Posterized Image] | ![Effect Image] |

For all pixels:
- if \[\frac{\text{red} + \text{green} + \text{blue}}{3} < 128\] set RGB to 0 | <br>**if** \[\frac{\text{red} + \text{green} + \text{blue}}{3} > 128\] set RGB to 255|
### Color Manipulation with Conditionals (Continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply an effect to part of an image by using conditionals to bound the x and y coordinates of every pixel.</td>
<td><img src="image1" alt="Conditionals Code" /></td>
<td><img src="image2" alt="Effect Image" /></td>
</tr>
</tbody>
</table>

### Interactive Color Manipulation Using Other Scratch Programming Blocks

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input mouse x and mouse y coordinates to interactively and continuously randomize pixel colors.</td>
<td><img src="image3" alt="Interactive Code" /></td>
<td><img src="image4" alt="Interactive Screenshots" /></td>
</tr>
</tbody>
</table>
V. USER STUDY WITH CHILDREN

Goals

Before the blocks supported conditional statements, they already provided a wide enough range of applications that their use could be tested with children in a user study. There were several aspects of the project that I wanted to test with this user study:

- the ability to make basic image-processing concepts accessible to children
- ease of use
- the children’s desire to apply the concepts in personally meaningful and exciting ways.

I knew that children were familiar with popular image effects from movies and video games that are commonplace today, so I decided to present the experiment as a new and exciting way to make these types of image effects on a computer. I hoped that this would tap into their interests and provide motivation. I decided to give them the specific task of creating a scene by generating different image effects for three objects. The idea was that they would have enough direction to concentrate on using the image filter blocks, but still have enough freedom to choose objects meaningful to them and be motivated to create the filters.

If I could get children to use the blocks for their own personal enjoyment and master the concepts enough to create with them, then I would feel that work in this area is useful and worth thinking about for the future of children and programming.

User Profile

To test out the blocks, I first had to pick a set of children that would suit the experiment. Ideally, the blocks would work with any children or teens who are learning to use computers as expressive tools. This is similar to the set of children that the Scratch programming language is geared towards. Many of these children are inner city youth that are going to Computer Clubhouses to learn to use computers for creative purposes.
However, because the image-processing blocks are an extension of Scratch, I felt that it would be better to test the set of blocks on children who had already learned how to use Scratch and who use it in their daily lives as a tool to express their ideas. I did not want to focus on teaching the users the concepts behind programming in Scratch, I instead intended to focus on the underlying image-processing concepts and their use. Therefore, to find users for the test I contacted a local teacher who had been using Scratch with her own two children at home: John, 11, and Mary, 9.

John had extensive experience with robotics and was in the process of entering a local robot competition with children his age (and older) in his local community. He was very involved in the robot competition, but he was also using Scratch at home on a daily basis. I had previously seen videos of him teaching other children how to use Scratch, which meant that he was comfortable with programming concepts enough to make them his own and share them with peers. Mary, 9, had experience with Scratch as well, and like her brother had been using it at home to make her own projects. Both children also used Adobe Photoshop regularly for photo cropping, resizing and also for artistic purposes.

These children were not typical for the kinds of children who would be using Scratch or who would ultimately benefit from the image-processing blocks, but I felt that they would be useful test users to see if the blocks could be used by children at all. If the image processing blocks were successfully used by these children, then it would be feasible to try to make them work for children with a less technological background.

**Presentation and Preparation**

Even though the users that were found were well versed in Scratch and general programming concepts, as I began to prepare the experiment I realized that there was background in simple image processing and digital imaging that I would need to teach them in order for the blocks to make sense at all. This realization was a slight setback because, ideally, the blocks should be learnable through exploration and intuitive interaction alone. However, I decided to present the formal background to image processing to the children as an exercise to see how much background was really needed and what amount of information would be enough, if at all, to use the blocks. The presentation covered the following
Other preparation included making sample projects with simple filters that the children could use as teaching models and inspiration. The samples contained simple color-changing algorithms along with algorithms that used mouse interaction, control loops and mathematical functions. I also included an example (refer to Table 5) that illustrated the type of scene that I wanted them to make during the experiment. The example scene contained two popular movie characters and one background image. Each object was altered with a distinct image-processing script for an interesting overall effect.

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three costumes were altered with simple image filter scripts to create a scene</td>
<td>Background&lt;br&gt;for all pixels&lt;br&gt;set RED to 1&lt;br&gt;set GREEN to 1</td>
<td>Original</td>
</tr>
<tr>
<td></td>
<td>Left character&lt;br&gt;for all pixels&lt;br&gt;set RED to 255 - red&lt;br&gt;set GREEN to 255 - green&lt;br&gt;set BLUE to 255 - blue</td>
<td>Result</td>
</tr>
<tr>
<td></td>
<td>Right character&lt;br&gt;for all pixels&lt;br&gt;set BLUE to 255</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Example scene which illustrates three objects with distinct image-processing scripts.
In addition to the presentation and the sample projects, another preparation step was to remove the native Scratch blocks that did not fit into the image-processing context. The only blocks not removed were control-structure blocks, mathematical blocks and sensing blocks for use as numerical arguments. Because support for conditionals had not yet been implemented, all conditional statement blocks were removed as well. I wanted to get a sense of what the children could create with just the basic set RED, set GREEN, set BLUE, set RGB, red, green, blue, x and y blocks along with basic mathematical functions and control structures. One important note is that I decided not to use the 0-100 range but rather the 0-255 range during this user study to see if the children would be comfortable with those numbers.

*The Presentation*

Before the presentation began, I asked the children if they knew what image effects were. They said image effects are ‘like what you see in movies or on TV programs, like CSI’. This confirmed what I believed about the exposure of children to image effects in everyday life. I showed them examples of image effects from photo-manipulation software, movies and video games to give them a concrete context in which to think about the concepts they were about to learn. These examples all seemed to resonate with them. I then introduced the concept of pixels as the makeup of digital images. I showed them concrete examples of pixels by zooming into a digital image on the screen. After seeing that demo, the shared with me that they had seen pixels before while manipulating images on computers, but had not known what they were called.

This led me to a discussion about the composition of light and how it affects the colors of a pixel. I discussed the color theory they already knew and presented the mixing of primary colors. They said they were familiar with primary color mixing from their schooling. This led me into teaching them about mixing colors on screen. To help present this concept, I visited an online applet which teaches basic color theory. This applet lets you interactively mix colors with three tubes of paint (cyan, magenta, yellow) and also with three lights (red, green, blue). Figure 6 shows screenshots of the applet.
Finally, I introduced the image-processing blocks as commands to do the color mixing which had just been demonstrated with the applet. I went over a few simple examples of how to use the blocks directly in Scratch. At one point I asked, ‘If I use set GREEN with argument 2*green, what do you think will happen to the picture?’ John quickly responded, ‘Well, it will turn twice the amount of green.’ John even went ahead to ask me ‘If I wanted to changed just a part of an image, how would I do that?’ This question confirmed that he had understood the notion of pixels and their manipulation, and at this point both children were prepared enough to use the blocks.

The Programming Session

When the experiment finally began, the first thing I noticed was the both children had very different working styles. John seemed to have a clear plan in mind and headed straight for a search engine to find the three graphics that he needed for the background and two sprites in his project. Mary just took the first image she could find with a quick online search and began experimenting with it. Once they both had objects to manipulate, they began their task of creating image filters for them.

The children seemed to have no problem understanding that they had to use the for all pixels block and put set RED, set GREEN, and set BLUE blocks inside of it to make changes to the pixels, and they were even comfortable using the 0-255 value range. Mary started out by changing one of her sprites to have a certain level of green, but I noticed that she also thought she had to set RED and set BLUE to
the original red and blue. It was not clear to her that that was not necessary. John started by using randomization blocks to set the three colors of the background. At first, he used the randomization blocks for all three colors of every pixel, but that gave a noisy image that was too random. Then, on his own, he decided to only randomize two of the colors (red and green). This gave results that he was more satisfied with since it was possible to make out some of the detail of the background image. He then decided to make the background dynamic by putting his randomization script in a forever loop.

During the programming session, the children were generally comfortable using plain numerical arguments, mathematical function arguments and the red, green and blue accessor blocks to set the color values of their pixels. Neither child used the x and y blocks for gradient effects, however. Both John and Mary made all of their scripts interactive or dynamic. John’s scripts used forever loops, mouse x, mouse y and keyboard input. Mary also used mouse x, mouse y and forever loops.

Some confusions and questions also arose during the programming session. The first question came from John who asked, “What would happen if I just ran the set RED to 255 block by itself without putting it inside the for all pixels loop?” I told him to just try it to see what would happen. When he tried and nothing happened, he asked me why. I explained to him that the image-processing blocks only made sense in the context of a for all pixels block, which he then told me he thought was unnecessary. He believed that the block should just work when executed by itself. His claim was ‘Well it should just work, you don’t really need the for all pixels block.’

A few confusing issues regarding the state of the pixel colors came up during the children’s explorations. One phenomenon happened when John used division in one of his image filters. He set all three colors to \((255/\text{original color})\) and, as a result, the image would turn to black and then back to the original colors repeatedly, because of reciprocity. Although he liked the strobe effect and was happy to have discovered it on his own, the functionality was mysterious to him. He did not understand that the colors were being set to reciprocal values and therefore just switching back and forth between two states. In general, it was difficult for the children to keep track of the state of the colors during mathematical manipulation.
**User Results**

In the end, both children successfully completed the task of creating a scene with three objects with distinct image-processing scripts. The overall themes of the projects were tied to interests of theirs, Napoleons and disco. Aside from the themes in each project's content, there was also a theme in the image-processing scripts that the children made. All six scripts that the children wrote involved user interaction. They were all dynamic, not just automatic like most of the filters which I had envisioned would be written with the blocks.

*John’s Project: Napoleons*

John's project had two Napoleons in it, Napoleon Dynamite and Napoleon Bonaparte. For his background, he chose an image of a castle. The script for the castle had randomization in it, only for red and green, and he used `mouse x` and `mouse y` user interaction to control the range of the random values. For Napoleon Dynamite, his script involved keyboard interaction. When the space bar was pressed, the character cycled between a negative image (since all its colors were set to the complementary colors) and the original image. When the space bar was let go, the image changed back to the original costume. The Napoleon Bonaparte object had the strobe effect discussed earlier because he continuously set the colors to `(255/original color)` which caused repeated switching between black and the original costume. The table on the following page displays John’s objects and the overall scene he created.
A scene with three image-processing scripts. Two were controlled with user input (keyboard and mouse) and one was applied dynamically in a forever loop.

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background</strong></td>
<td><img src="image" alt="Background Code" /></td>
<td><img src="image" alt="Original Image" /></td>
</tr>
<tr>
<td><strong>Napoleon Bonaparte</strong></td>
<td><img src="image" alt="Bonaparte Code" /></td>
<td><img src="image" alt="Interactive Results" /></td>
</tr>
<tr>
<td><strong>Napoleon Dynamite</strong></td>
<td><img src="image" alt="Dynamite Code" /></td>
<td></td>
</tr>
</tbody>
</table>

*Mary’s Project: Disco Creatures*

Mary's project evolved during the programming session, but in the end she converged on something with two creatures, a disco ball and a hand-drawn disco floor. She did not add an image-processing effect to the floor because she wanted the disco ball in the room to change but not the whole disco room. The disco ball image effect had randomization of every pixel channel in a forever loop to make it look like it was glittering. The creatures’ costume colors were determined interactively by `mouse x`.
and mouse y, setting different colors with different parameters for each creature. In the end, she created a dance scene where the disco ball was continuously flashing different colors and then the user could determine what colors the dancers were lit up by moving the mouse around. The following table shows Mary’s final scene, objects and scripts.

<table>
<thead>
<tr>
<th>Description</th>
<th>Blocks</th>
<th>Effect</th>
</tr>
</thead>
</table>
| A scene with three objects, a glittering disco ball using a forever loop, and two creatures that changed color according to mouse input. | **Disco Ball**

```
for all pixels
set RED to pick random [0 to 255]
set GREEN to pick random [0 to 255]
set BLUE to pick random [0 to 255]
``` | **Original** |

<table>
<thead>
<tr>
<th><strong>Left creature</strong></th>
<th><strong>Interactive Results</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forever</strong></td>
<td></td>
</tr>
<tr>
<td>for all pixels</td>
<td></td>
</tr>
<tr>
<td>set GREEN to mouse x</td>
<td></td>
</tr>
<tr>
<td>for all pixels</td>
<td></td>
</tr>
<tr>
<td>set RED to mouse y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Right creature</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forever</strong></td>
<td></td>
</tr>
<tr>
<td>for all pixels</td>
<td></td>
</tr>
<tr>
<td>set GREEN to mouse y</td>
<td></td>
</tr>
<tr>
<td>for all pixels</td>
<td></td>
</tr>
<tr>
<td>set RED to mouse x</td>
<td></td>
</tr>
</tbody>
</table>
In the end, I asked the children what they thought of the process. Their comments to me were that the activities were 'pretty simple' and 'fun'. The mother exclaimed that the blocks were a great way to demystify computer graphics and the type of work that ‘Pixar does.’ The children were happy to be using graphics that they could resonate with, like Napoleon Dynamite and fantasy creatures at a disco. They found the blocks easy and fun to use for creating new and interesting image effects.
VI. CONCLUSION

Overview

The goal of this work was to create a programmable toolkit to expose children to basic image-processing concepts and empower them to create new and interesting artistic image filters. The Scratch programming environment was the perfect context for this work because it provided a media-rich programming environment for children with room for an image-manipulation toolkit. Through the implementation case studies and the user study, it was shown that the toolkit was successful in bringing basic image-processing concepts into the hands of children so that they were able to create new and personally meaningful image effects with them.

However, there are flaws in the toolkit design that make it less than ideal for use within Scratch. Also, some more work needs to be done in order to create an image-processing toolkit that can be used without needing to present background information in advance. In the following sections, I present the factors that contributed to the success of the system and also the issues that arose which suggest that the toolkit needs further work. In general, this work demonstrated that future research in this area is worthwhile.

What Worked

Scratch as a Context

One success of the toolkit design was the decision to use the Scratch programming environment as a context. Scratch offered a great model for programming because of its graphical-block language which children seem to learn more easily than syntax-heavy text-based languages. Working in this context also provided the opportunity to test the image-processing blocks with children who were used to this programming language and did not need to learn it in advance. Also, adding special cases to the Scratch execution model to handle the new image-processing blocks was straightforward since the infrastructure was already set up for other command blocks.
During the user study, the children themselves seemed to latch on to the power of Scratch as an interactive programming tool because they didn’t just create static image effects but introduced user input and dynamic features to their image filters. This ability to create dynamic filters came from the other programming blocks in the Scratch environment. Even though it was not the focus this work to create interactive filters, it was interesting to notice that the children took the image-processing blocks and combined them with even more tools within Scratch to achieve their envisioned effects. The interactive element of Scratch was a perfect complement to this work as the children were able to create image filters which I had not even envisioned.

*Concepts Successfully Exposed*

The presentation during the user study introduced some of the image-processing concepts to the children before using the blocks. For example, the children had never formally learned about pixels. The only previous exposure they had had to pixels was from manipulating images on computers and zooming in. From the presentation, however, they were able to make the connection that they could change the pixels to create image effects.

Even though the concepts were exposed by the presentation and not the blocks themselves, the blocks did make the concepts accessible and tangible to the children. Directly after the presentation, the children were able to use the *for all pixels* loop and set the red, green and blue for each pixel to change color values. This showed that the children immediately understood the mechanics of changing pixels and their light composition as soon as they sat down to use the blocks. The simple metaphor of the abstract *for all pixels* loop and the accessors and modifiers made sense to the children. This demonstrated that fundamental, image-processing concepts were successfully uncovered by the toolkit.

The children’s remarks at the end of the programming session that the exercise had been ‘pretty easy’ was also evidence that the concepts were made accessible by the blocks. In addition, the distinct image effects that both children made are proof that the concepts were then applied in innovative ways.
Engaging Material

From the user study, I gathered that the children not only learned how to use the blocks but also enjoyed what they were doing as they personalized their projects and customized their filters for their specific purposes. They applied the concepts to objects they cared about, which helped fuel their decisions about which effects they created and which blocks they used to help them get there.

John used the randomization blocks to generate the colorful effect that he envisioned for his background. He also wanted user interaction to control his Napoleon Dynamite character, so he included keyboard input to cause the character to change color. Mary wanted an exciting disco ball for her scene, so she used the randomize blocks to give the effect that the disco ball was glittering. Then, she wanted her characters to light up to look like they were in a dance hall, so she had them change color interactively with user input.

As mentioned in the introduction, this project was about combining the best of both worlds, programming and digital artistic expression. The latter part is closely tied to content and personal interests and the user study confirmed its importance. If the children had not found motivating images that were appealing to them, they would not have had the motivation to use the image-processing blocks in interesting ways.

A Feasible Activity

During the programming session, the children learned, asked questions, and understood the blocks enough to use them to create their own image filters. Given the users I had picked, however, I am not surprised that the programming session went smoothly. The children were bright and had been using Scratch daily and, most importantly, were already used to using it to program their own artistic creations. However, the user study confirmed that using the blocks is feasible for children of ages 9-11, which supports further research in the area.
**What Didn't Work**

From the implementation of the blocks to their use in the user study with the children, I encountered some problems in the design that I had not anticipated. The major flaws of the design were that the use of the blocks required presentation to teach the fundamentals of image processing, the concepts exposed by the blocks raised new and confusing issues, and the blocks did not fit 100% naturally into the existing Scratch programming model.

**The Necessary Background**

One flaw of the blocks’ design was that they required presentation before being used with children. Ideally the blocks would be self-explanatory and a child could just experiment with them and, through their use, understand how they work. However, before running the experiment with the two children, it was clear that there were some core terms and ideas that they would need to learn before using something such as a **set RED** block. A block like this might cause a child to ask: What red are you talking about? What does the numerical argument represent? How does one block change the whole image? What are the parts of the image that this block is affecting? Although using the image-processing blocks helps answer some of those questions, it was difficult to make full understanding come about without some background in digital imaging and basic image processing.

This necessary background raises a question as to whether this framework would be an asset to a programming language where most blocks and commands are discoverable through exploration. Perhaps if a child encountered a **set RED** block and tried to run it, he/she might immediately see the result, much like with other Scratch blocks, and this visual feedback would start to give some sense of what the block is affecting. However, it still might not be clear that an image is made up of three independent color channels, and that gap in understanding would be a big hindrance to understanding the rest of the image-processing blocks.

However, this experience did illustrate that the theory of additive color is an important one to start teaching children. In this new age where everyday media is becoming digitized, children will be constantly exposed to RGB video displays as the paint canvases of everything they see. Image-
processing concepts will become tools that children will use to make their artistic creations; so teaching children about the fundamentals of digital imaging, just as was done in the user study, is worthwhile.

Confusion with Concepts Exposed

Although the blocks exposed underlying image-processing concepts and gave children the power to create their own image filters, there were some unexpected and confusing issues that arose with their use in the user study. Explaining many of these concepts to users would take even further background, more than is feasible for a system where a child is supposed to be learning by exploration.

The most confusing issue that arose dealt with the state of the pixel colors during execution of a single script. When designing the blocks, some assumptions (or careful choices) were made about how to represent the state of the colors behind-the-scenes. One choice was that if a pixel value was incrementing and reached the value 100, then it would stay at that value and not wrap around. Or, for example, if a child set a color channel to 300, it would act as if the color channel were clipped at 100. Similarly for negative numbers, the channels were clipped at 0. This was done to eliminate the confusion of trying to set a color value to one less than zero but then see the color value abruptly jump to 100. However, as the children used the blocks during the session and got surprising and mysterious effects, I realized that this simplification was still not enough to make the color value range clear. The blocks are not transparent enough that a child can see the value and state of colors as they change.

Another confusing issue regarding the state of the colors came from the red, green, and blue reporter blocks. As designed, the red, green and blue value reporters always report the initial values of the current pixel. Instead of dynamically being able to change the red value of the pixel and use it later in the same execution loop, it was made so that the red, green, and blue reporters always report the original value. This was another choice made for simplicity, so that a child could always use the original values and not be confused when trying to access an original channel of the pixel twice. However, if a child tried to write a script where they first set red to 0 and then tried to set green to red and expected the value in green to be 0, it would instead be the value of the original red level. This is a hidden and potentially confusing feature of the blocks that I had not anticipated.
Fixing these confusing issues would take a more careful design of the blocks. However, this experiment helped to successfully test the feasibility of the blocks and to expose the true issues that might arise, which in turn helps to make more careful decisions about what concepts to expose.

*The Incompatibility with Scratch*

Early on, I realized that the blocks did not fit naturally into the Scratch language. The Scratch programming metaphor and current set of blocks is designed in such a way that all of the blocks are independent and can be snapped together in any way. Any reporter can go into any argument slot, any block can go into any C-block script, and all of the block stacks can be stacked interchangeably. The image processing blocks, on the other hand, are designed specifically for use on the pixels of the sprite’s current costume. For example, it is not possible to use a ‘motion’ or ‘sound’ block inside the *for all pixels* loop. Likewise, it is not possible to put the *red/green/blue/x/y* reporter blocks into the argument slot of any other command block in Scratch that is not *set RED*, *set GREEN* or *set BLUE*.

Also, if you tried to execute one of the *set RED*, *set GREEN* or *set BLUE* blocks by themselves, then Scratch returns an error. Basically, the only integration that I offered is that an image-filter script could be stacked onto any other script in Scratch. However, you could not break apart the script itself and use the blocks in other contexts in Scratch because the blocks are conceptually separate from the rest of the Scratch programming environment.

If I wanted to include the blocks into the actual Scratch language, then another evaluation metaphor that was more in tune with the current Scratch blocks would need to be developed. Even though this was a setback early on in the project, I decided to keep implementing the design because it was still interesting to see how the blocks and the concepts worked with children in the context of a limited, yet still helpful, Scratch language.
Final Conclusions

The successes of the project were encouraging and, although there were setbacks and outcomes that did not fit in with the original expectations, they were not discouraging enough to invalidate the work. The image-processing blocks from Scratch were demystified to a point where the children understood how to create their very own image-processing algorithms.

Is it Useful to Teach Children Image Processing?

In this project, children went from not knowing what pixels were called to being able to generate their own innovative image filters. It was inspiring to see that children could potentially learn the fundamentals of image processing, which they are slowly becoming more and more exposed to each day, and this toolkit seems to be a good starting point for this kind of education.

The toolkit was also a useful learning tool because, with it, children can learn about the general concepts of digital art and additive color mixing. However, this work also showed that the image-processing concepts needed introduction and preparation and that it is not always easy to understand the concepts without guidance.

Ken Perlin gave feedback about this work and noted that, essentially, the children using the image-processing blocks would be creating 'shaders', image filters which affect only the color of every pixel of an image. In the real world, shaders are use in everything from computer graphics animations to movies, video games, television commercials and artistic software. The concept of shaders is something children encounter everyday, so it is definitely a concept worth teaching.

A Separate World from Scratch

It was useful to teach children image-processing concepts, however, the fact that the blocks did not fit naturally into Scratch shows that this type of work would be best explored in its own context. The concepts are interesting and separate enough that separate work in this area could be interesting on its own. While Scratch was a great avenue to explore this type of work because of the programming
model that it provided and the conceptual motivation of combining programming and artistic expression, a stripped down version was still necessary for the user study and the image-processing blocks were not compatible with all Scratch commands. A more suited setting could perhaps be a ‘microworld’, [5] or focused digital artist studio, where children are constantly exploring this area of customized image filters.

This work shows that children can program computers to create their own, innovative digital art. Instead of using traditional photo-manipulation software which has built-in tools for them to use, with work like this, they can understand what is going on behind the scenes and think about the process that goes on when manipulating images on a computer. This concept is powerful because it means children can achieve the effects that they want, be imaginative and learn about computers all without being bound by what someone else has already defined.
VII. FUTURE WORK

This project opens up an area of future work that exposes children to the programming of images on the computer. This leads to learning how to use a computer as an actual paint brush for things such as non-photo-realistic rendering, image shaders, computer graphics and computer animation, all of which are quickly becoming an integral part of modern media. In contrast to programs such as Macromedia Flash, Adobe Photoshop and KidPix, work like this could make children their own creators and inventors of image-processing techniques that some adults might never envision. Also, with so much youth interest in digital photography, children nowadays have plenty of inspiration to come up with personalized, fun image effects.

Some specific ways in which this work can be expanded in the future include:
- making it more integrated into Scratch so that it actually becomes a tool in Scratch that a child would use alongside the other creative commands and blocks
- adding more functionality to the current set of blocks
- using this teaching model in introductory computer science and media computation classes or
- putting this new language into its own module of introductory image processing.

Improve Compatibility with Scratch

If the blocks were to be included into the actual Scratch language, then another programming metaphor that was more in tune with the current Scratch blocks would need to be developed. The blocks would need to be able to run standalone and other Scratch blocks would need to be able to fit into the for all pixels block. The three major changes would be, fix compatibility with hat blocks, make standalone blocks work and make for all pixels more compatible. Conceptually, the blocks would need to be more compatible with the current image-filter blocks in Scratch, although it is also a question whether those would just be replaced. Another possibility is to create a separate image-processing mode in Scratch where children would go to program their own filters and then store those filters into an image-effect block that could then be used like the rest of the image-filter effects in Scratch. For this to be successful, it might make more sense to include a general behind-the-scenes function creator in Scratch that let children create their own blocks of any type.
Also, to help with the integration of the blocks into Scratch and to help make the blocks more intuitive, their design could be changed according to the feedback from the children during the session. For example, one thing that could be changed is the **for all pixels** block. According to John, he felt that the block was not necessary in order to achieve the desired effect. For example the **set RED** block could just incorporate the **for all pixel** evaluation mechanism inside of it, removing the need for the **for all pixel** block altogether.

**Add More Features**

Adding more features to the blocks would involve adding more commands to the language and more image-programming concepts including: Perlin Noise, variables, and geometry for distortion filters. Perlin Noise features would allow creation of more natural and generative image effects. Adding variables into the language would give greater control and more flexibility of the numerical values passed as arguments. The distortion filters that are possible with more geometric control would be a great asset because they are generally appealing and interesting. However, these new features have to be designed in such a way that they are easy to understand and do not require as much background as the current set of blocks. Another idea is to make the blocks literally transparent enough that kids are actually seeing the color channels in every pixel change (in a color chart or color bar) so that they understand how the colors are affecting each other and how their blocks and scripts are ultimately affecting their images. This could be engaging and would really take opening the black box to a new level where there is pure visual feedback as to what is going on inside the image filters.

**Incorporate into Introductory Media Computation Courses**

Another possibility for future work is that these blocks could be used in the context of courses like Guzdial's,[4] which introduce programming to novices through media computation. These blocks could be useful in those classrooms because they go one step further in simplifying image-processing concepts by taking away the most of the syntax needed, a feature which is owed to the Scratch programming model. If Guzdial's class were transferred down to the high school level or middle school and even elementary school level, these blocks would be an ideal merger of Guzdial’s work and a young student’s level of expertise with computers.
Create a New World: The Best of Both Worlds

Creating a separate piece of software where children explore the basics of image-processing to make their own image filters would not be a replacement for software such as KidPix, Adobe Photoshop or Scratch. Rather, it would be a complement where curious minds could go to learn image processing and to play with the notion of creating their own image effects. As this worked showed, given the right tools, children’s imaginations allow them to create interesting and innovative artistic digital effects which come from their own understanding of computers and art. This kind of digital art education can give children the power to use computers to not just view art, but to create it too.
REFERENCES


APPENDIX A.

Using Scratch to Make Your Own Image Effects

What are Image Effects?

- Change a picture to look like something else
  - Color, Shape, Lighting, Lines, anything!
APPENDIX A.

Image Effects in Movies

- Games
- Photoshop Filters
What You Will Do

- Use Scratch to create your own image effects
- Only color image effects
- Not the same Scratch as before
  - Fewer blocks: no motion, pen etc..
- Help me test the blocks
  - Testing the blocks, not you!

What Are Images Made Of?

- Images are made of little squares called **pixels**
  - Each pixel is just one color
APPENDIX A.

How Are Images Changed?

- To change an image, change all of its pixels!

How Are Pixels Changed?

- To change pixels, change their color
- Pixels are made by mixing three colors:
  - Red (0 - 255)
  - Green (0 - 255)
  - Blue (0 – 255)

- http://www.omsi.edu/visit/tech/colormix.cfm
APPENDIX A.

Image Effects in Scratch

- Use Scratch blocks to change all of the pixels in a costume
- Blocks:

```
for all pixels
  set RGB to 0
  set RED to 255
  set GREEN to 50
  set BLUE to 255
```

- Don't have to use just numbers

```
for all pixels
  set GREEN to 2 * green

for all pixels
  set RED to x

red
green
blue
x
y
mouse x
mouse y
```
APPENDIX A.

Goals

- Create your own scene with image effects
  - Pick at least 3 images from the web
  - Change them in different ways
  - Show me what you learned

- Have fun! 😊