Exploring Real-time Video Interactivity with Scratch

by

Ting-Hsiang Tony Hwang

S.B., E.E.C.S. M.I.T., 2007

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 2012

Author: _____
Department of Electrical Engineering and Computer Science
May 21, 2012

Certified by: _____
Prof. Mitchel Resnick, Thesis Supervisor
May 21, 2012

Accepted by: _____
Prof. Dennis M. Freeman, Chairman, Masters of Engineering Thesis Committee

Exploring Real-time Video Interactivity with Scratch

by

Ting-Hsiang Tony Hwang

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 2012

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

ABSTRACT

Real-time video interactivity is becoming increasingly popular in today's world with the advent of better and more affordable video input devices. With the recent release of the Microsoft Kinect followed by an official Kinect SDK, there has been an explosion of activity around utilizing this now easily-accessible video sensor data. Many creative uses have surfaced including object recognition, gesture recognition, and more. The audience capable of taking full advantage of these video technologies continues to be a technical crowd, likely with a background in computer science. But what if such video technology were made accessible to a much more diverse crowd?

This thesis presents a set of computer vision tools for exploration of the real-time video interactivity space in the context of Scratch (scratch.mit.edu), a graphical block-based programming language accessible to all ages. To decide what functionality to provide to Scratch users, various computer vision algorithms are tested, including object detection, object recognition, face recognition and optical flow. Ultimately, an optical flow implementation is realized and its generative abilities are observed through testing with different user groups.

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

ACKNOWLEDGEMENTS

It has been a fulfilling year of learning and development, academically and personally. I would like to thank the following individuals for making this experience possible for me.

- Mitchel Resnick for generously enabling me to pursue my research interests in a safe environment, and allowing me to contribute to the amazing ecosystem of Scratch.

- John Maloney for always making time to share his programming expertise with me with great patience and care.

- The MIT Media Lab's Lifelong Kindergarten group for making me feel welcome and opening my eyes to the importance of community.

- My parents, Ar-Huey and Chorngjen, for keeping my best interests at heart and supporting me from afar.

- Lucy for truly believing in me when I most needed it, and encouraging me to find my way.

CONTENTS

1. Background

One of the major goals of this thesis was to make accessible computer vision capabilities to anyone interested, not just programming-savvy technology enthusiasts. An excellent medium through which to reach such a wide audience is Scratch. Developed in the Lifelong Kindergarten Group (LLK) at the MIT Media Lab led by Prof. Mitchel Resnick, Scratch is a graphical programming language that embraces the design philosophy of "low floor, wide walls, high ceiling" [1]. The "low floor" means that Scratch is easy to learn, and one can very quickly produce enjoyable project results. A conscious decision was made to do away with error messages that could discourage the user, and instead imbue each block with intuitive behavior in any situation that best conveys the essence of the block's functionality [2]. The "wide walls" describe the wide range of functionality that is made accessible to the user. Scratch provides many tools that allow the user to manipulate various kinds of media. Finally, the "high ceiling" indicates that in trying to create projects, users should only be constrained by their own minds, not by the Scratch tool itself.

Since Scratch's release to the public in 2007, the community has grown rapidly such that more than 2.5 million projects have been shared, with thousands more being shared daily. Although the typical user may not be aiming to learn about computer science or computational concepts, through pursuing projects of their own interest, they are indirectly learning and becoming more confident as creators. Thus, Scratch seemed like a natural choice for a powerful but convenient environment for users of all levels of technical aptitude to explore the creative possibilities with computer vision in a friendly and forgiving space.

It is now convenient and affordable to acquire a webcam and to set it up to view/record video. However, a machine does not understand how to parse the video as humans can. Computer vision is a field that seeks to expand the machine's capability of interpreting images and video. This field is experiencing rapid growth as visual media content becomes increasingly present and important in our lives. One specific area that is still being explored is real-time video processing. In this thesis, I seek to abstract away the technical details of computer vision algorithms from the user, and give easy access to the functionality. By providing users a convenient window into a technical but important area of research, I hope that many minds will be opened to a new design paradigm currently only reserved for the few.

2. A Brief Survey of Computer Vision Techniques

In order to understand the potential of video-based computer vision techniques, it is instructive to view some examples that can help map out the current capabilities. Computer vision (CV) involves the processing, analysis, and interpretation of images and video to draw meaning from such visual media. Examples of CV algorithms on static images range from using optical character recognition techniques to understand text that appears in a photograph to estimating the position of the dominant light source in a scene. CV algorithms that are used on video often focus on tracking movement through time. For example, a video surveillance camera stream could trigger a notification when abnormal movements are spotted. It could also analyze the swing trajectory of a golfer.

With such a wide range of possibilities, there were many candidates for algorithms that could be added to the Scratch block vocabulary. The following sections contain further depth about the trial and decision process for choosing computer vision algorithms for Scratch.

2.1 Previous Experiments with Video in Scratch

Video has not been a supported data type in released versions of Scratch. In the original Scratch design discussions, a conscious decision to opt out of video support was made because of the fear that video clips would make Scratch projects more "linear" than interactive. However, as webcams became more prevalent and affordable, video input could be considered in real-time, and experimentation with video resumed.

One significant foray into the video space in Scratch was a research project called Color Code [3], developed by LLK graduate students Eric Rosenbaum and Jay Silver. The Color Code blocks allowed a user to filter webcam video to detect a specific color specified by the user. Specifically, it transformed the webcam video frames into the hue-saturation space, and would return the pixels that fell within a certain specified tolerance of the target hue. So, users were able to trigger actions for their sprites based on if they were touching the target color. Color detection was a very generative idea and the first time that the webcam input was processed in Scratch. I hoped to build upon this foundation and investigate further possibilities.

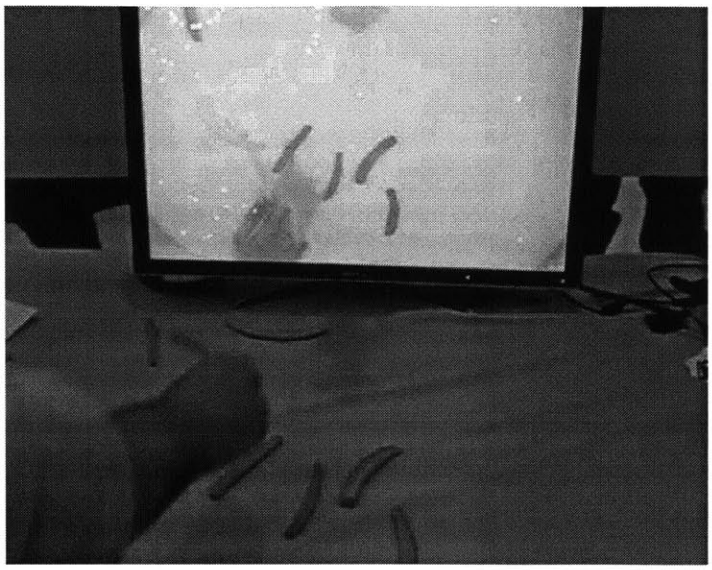

Figure 1. Color Code alpha demonstration. A webcam is trained on the table surface, and the objects seen are filtered by color.

An extension of Scratch (but distinct from the main branch) that uses a webcam is AR SPOT [4]. AR SPOT extends the Scratch functionality to support webcam detection of known QR codes printed on cards. Given knowledge of these pre-set QR codes, the application can determine the position and orientation of the codes, and use this information to interact with sprites. I wanted to try to devise a system that could operate without calibration against known objects.

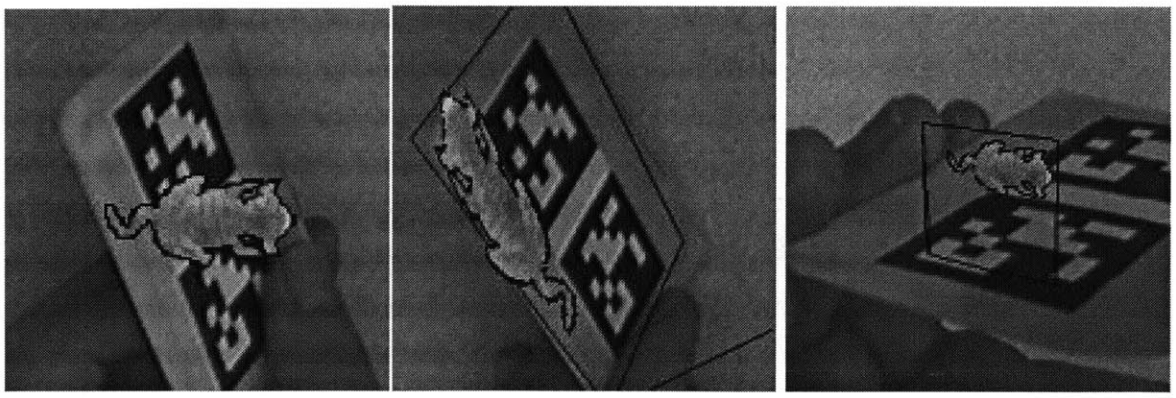

Figure 2. AR SPOT functionality. The images display the ability of the software to map position and orientation to a sprite.

One recent Scratch-based extension that made use of computer vision algorithms was called Vision on Tap [6]. This project demonstrated an implementation of computer vision algorithms as an internet service with a user friendly web interface. It used the Scratch block paradigm to control the algorithms, but used a separate application to handle all of the CV computation.

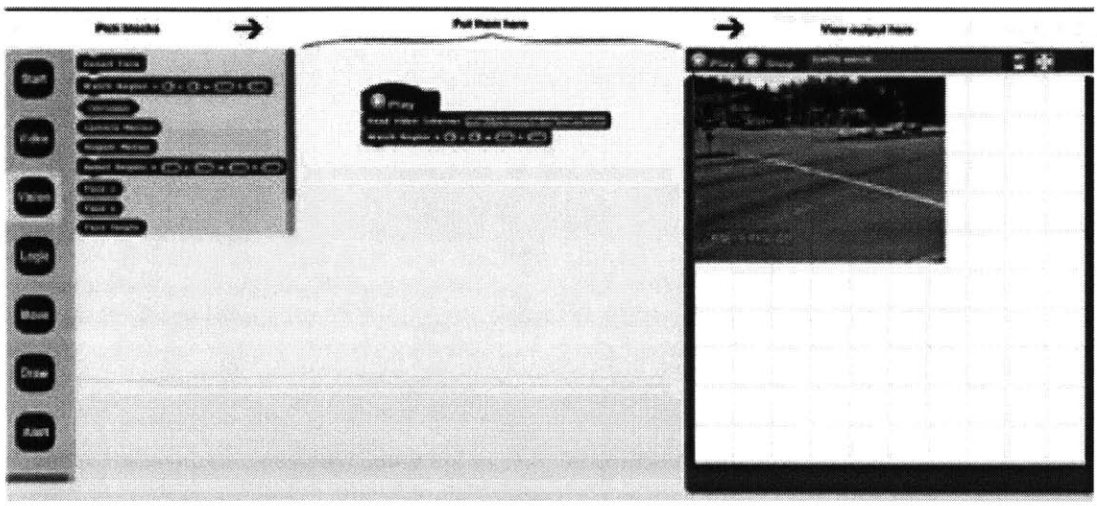

Figure 3. Vision on Tap GUI. This layout is similar in many ways to Scratch, but the blocks are designed solely for computer vision applications.

2.2 Potential New Computer Vision Algorithms for Scratch

2.2.1 Design Considerations

Given its media rich programming environment, one could easily desire to add many new features to Scratch. However, Scratch has been carefully designed with certain guidelines in mind. In trying to keep consistent with its design philosophy, as well as through experimentation, I was able to reduce the new features to its final form.

Scratch gives its users simple yet powerful tools. It aims to find the perfect balance between providing enough functionality to get the user started easily without depriving them of the creative process of developing their own complex applications. Accordingly, I sought to incorporate the simplest yet most generative and instructive CV functionality into Scratch.

Additionally, Scratch is intended to run on any computer and reach the most people. The new version (Scratch 2.0) will run entirely in a web browser. Traditionally, much of computer vision application

development is done in C++ because of its relative speed compared to other languages. Scratch 2.0 is written in Flash ActionScript, which runs significantly slower than C++ code. Thus, the CV algorithms would need to be especially computationally lightweight relative to the provided functionality. Also in keeping with the goal of reaching the most people, the CV algorithms would need to work with a webcam vs. a more complex and expensive video sensor (such as the Microsoft Kinect).

With these design criteria in mind, I began my feasibility research on potential algorithms. The following sections describe these algorithms in some depth.

2.2.2 Object Recognition

The first algorithm I looked into was object recognition. Object recognition involves comparing an input image against a known template image database and attempting to identify a match. One of the more recent techniques for object recognition involves the notion of feature matching. Unique features can be computed for each image, and if enough feature correspondences are found between two images, a match has been made.

One algorithm for feature computation is called Speeded Up Robust Features (SURF) [5]. SURF is a relatively fast way to compute scale-invariant and rotationally-invariant features in an image. It locates the positions of high pixel intensity gradients in the image by using the Hessian matrix, sometimes also known as the structure tensor. The Hessian determinant is used as an indicator of pixel intensity gradient value. If the eigenvalues of the Hessian matrix are of the same sign, the involved point is marked as an extremum for gradient value.

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x * I_y \\ I_x * I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} < I_x^2 > & < I_x * I_y > \\ < I_x * I_y > & < I_y^2 > \end{bmatrix}$$

Figure 4. Structure tensor or Hessian matrix.

The image coordinates that are found to have high brightness gradients are deemed SURF keypoints. A SURF descriptor is then computed for each keypoint. SURF descriptors are vector representations of the local neighborhoods around the key points. More specifically, a descriptor contains pixel intensity gradient information in the x and y directions, for a rectangular area centered at the involved key point. These gradients are calculated through the use of Haar wavelets. Haar wavelets are also used to ensure rotational invariance. Such properties of the descriptors make them very suitable for template matching.

Once we have a set of SURF descriptors for each potential template image, we form a vocabulary tree of these images using k-means clustering on the SURF features. This tree structure formed from the clustering is often called a bag-of-words (BoW) model. The BoW is built in the descriptor space, where proximity between features is determined by comparing their Euclidean distances. Specifying a positive integer k describes the number of feature clusters that will be formed. Once the BoW is built, the SURF descriptors from each template image are then "pushed" through the tree. Pushing a descriptor through the tree consists of finding which cluster it most closely belongs to based on Euclidean distance. After pushing all of the descriptors for an image through the tree, a set of counts associated with the clusters is recorded. This process is repeated for each template image.

Finally, the input image's SURF descriptors are pushed through the tree. This results in another set of cluster counts. The template image with the most similar distribution of counts is likely then determined to be the best match. Using k-means clustering and BoW is motivated by the drastic speed up in computation time compared to doing pair wise comparisons of each input descriptor with each potential template descriptor.

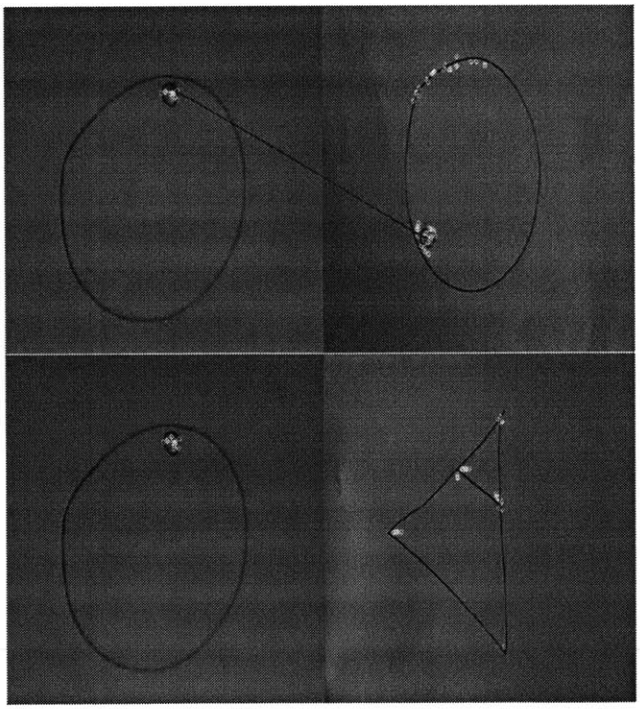

Figure 5. Object recognition results of a particular frame. The test image on the left has matched sufficient SURF features (indicated by yellow circles) with the circle from the vocabulary tree on the right, and not with the triangle from the vocabulary tree.

2.2.3 Object Tracking

The Condensation algorithm [8] is an application of the particle filter [7] to object tracking . It consists of an iterative process for each new video frame that alternates between using a prior model based on the system dynamics to predict the motion of the tracked object, and updating the model based on new measurements. This Sequential Monte Carlo approach is applied to a set of samples (particles), and a sufficient number of samples can then approximate a prior distribution, thus eliminating the need to have an exactly-specified prior distribution. The Condensation algorithm is thus well-equipped to deal with tracking objects through clutter or occlusion due to the flexibility of the model representation.

Each time step of the algorithm uses Bayesian probability to update the samples. Each sample is specified by a state vector x which can consist of any desirable characteristics to track about the object's motion. Typically, state vector elements are at least the position of the sample $\boldsymbol{x} = [x, y]$, but depending on the application might also contain velocity, rotation, acceleration, or other components. When the algorithm begins, it is assumed that the current measured state z_t and all prior states of the tracked object are known. It is important to note that knowing z_t is equivalent to knowing $z_0, z_1, \dots, z_t$ because of the Markov nature of this algorithm. By the end of the time step, we would like to compute the posterior probability density $p(x_t|z_t)$. Thus, by Bayesian logic, we need to compute

$$p(x_t|z_t) = p(z_t|x_t)p(x_t|Z_{t-1})/p(z_t|Z_{t-1})$$

During the prediction phase, we represent the prior $p(x_t|Z_{t-1})$ as the dynamics model. That is to say that the predicted position for the current time step will be based on our previous knowledge of the particles movements. The dynamics model update typically consists of computing a combination of the state elements as well as accounting for random variation (noise).

$$x_t = Ax_{t-1} + Bw_t$$

Once a sample's next state has been predicted, it is compared against the measured state $\pi_t = p(z_t|x_t)$ .This comparison can be based on any logically chosen metric. Evaluating this metric will yield a confidence in the prediction, and accordingly a weight that specifies the relative contribution of the sample to the contribution of the other samples. These weights must be normalized such that $\sum_n \pi_t = 1$, and that the cumulative probability is:

$$c_t = c_t(n-1) + \pi_t(n)$$

Now the posterior can be computed. The predicted samples become the samples for the next time step if there is high confidence in the prediction. If a predicted sample has low confidence, it can be reinvigorated through Sequential Importance Resampling (SIR). The low-confidence samples can be redrawn from a distribution near the higher confidence samples. Thus, the samples should eventually converge to the ones that best represent the object's motion.

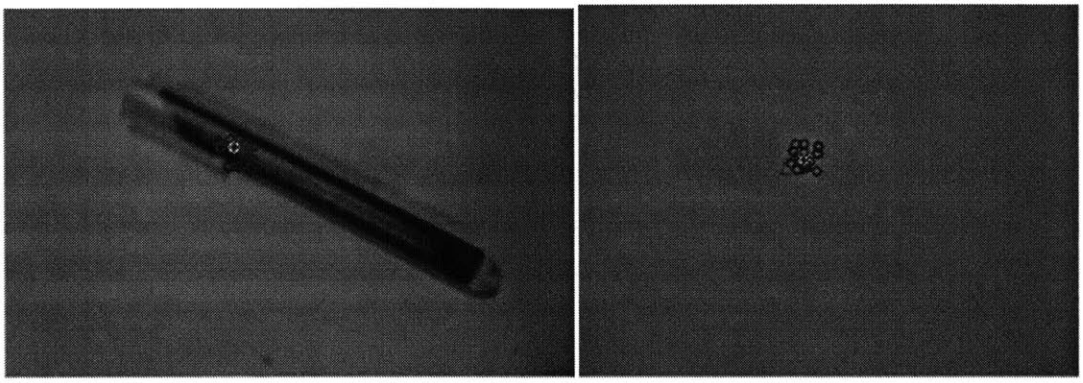

Figure 6. Screenshots of object tracking results. For each image, the blue circles represent the estimated centroid locations of all states. The green circle represents the weighted average centroid location.

2.2.4 Face Detection

Face detection is most commonly performed using the Viola-Jones face detection technique [12]. It relies on training a cascade of classifiers on a large sampling of faces, and then using these well-trained classifiers to later quickly identify faces. For the purposes of my experimentation, I focused on forward-facing faces, but there are other versions of the detector that detect other face orientations.

The training portion of the detector uses the AdaBoost boosting technique to train a set of weak classifiers. The training data is a database of thousands of different faces that have been normalized in scale and translation. These faces differ slightly in pose, lighting, and they belong to different people. The classifiers are based on Haar wavelets. After running the classifiers on the training set, eventually boosting converges on a set of weights for a set of Haar features that form a reliable face detector.

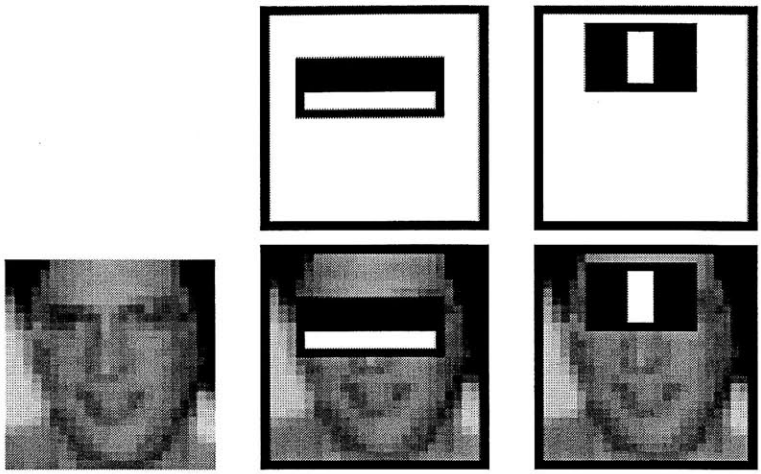

Figure 7. Training classifiers using Haar features. The two-dimensional Haar wavelet-based classifiers are shown as adjacent patterns of white and black rectangles. After training on thousands of faces, the classifiers are most efficiently positioned and weighted such that they capture typical brightness gradient changes on a human front-facing face.

Once the detector is fully trained, it can quickly be used on an input image. Haar features are quick to compute using integral images. Integral images are integrals of pixel data taken over rectangular areas of an image. By storing an integral image for all possible rectangles in an image, Haar features can be quickly computed as a sum or difference of these rectangles. The scale of any detected face can then be determined depending on the size of the features found.

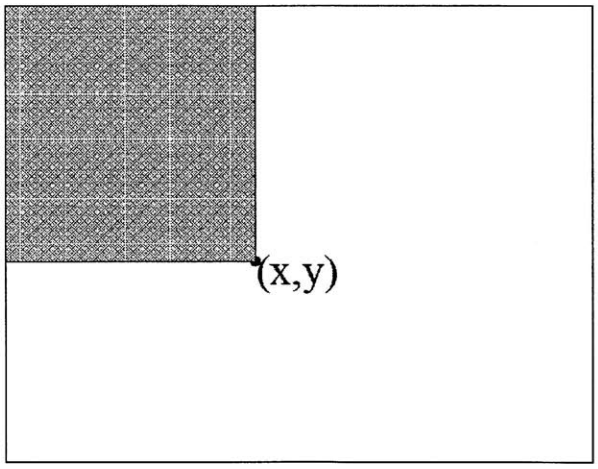

Figure 8. Integral Image. The integral image of position (x,y) is the integral of the rectangle up and to the left (shaded).

2.2.5 Optical Flow

Optical flow is the apparent motion that happens in a video sequence. In the popular Lucas-Kanade method of computing optical flow, differentials are used to estimate the magnitude and direction of motion in a local area. The algorithm assumes that the scene has a constant brightness, and that the displacement of moving objects between frames is relatively small compared to the size of the object. In order to preserve these constraints, the following equation must be satisfied for all pixels.

$$I_x(q_1) * V_x + I_y(q_1) * V_y = -I_t(q_1)$$

Figure 9. Brightness Change Constraint Equation (BCCE).

The optical flow equations for each pixel result in a large system of equations that can be rewritten as a matrix equality. Because the algorithm assumes that the motion in a local area is constant, the system is over-constrained, and thus least-squares can be applied to optimize the flow computation.

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_x(q_i)I_y(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

Figure 10. Computing horizontal and vertical velocity components.

The formula for using least-squares to compute the horizontal and vertical velocity components is displayed above. The optical flow of a pixel relies on the horizontal and vertical gradients as well as the gradient across time for its position. Because images are formed from discrete pixels, discrete gradient approximations are used to compute spatial gradients. One such discrete gradient operator is the Sobel operator.

The Sobel operator is a 3x3 matrix that is convolved with the target image. The Sobel operator is popular because it is separable in horizontal and vertical directions, and its convolution can be further decomposed into two simpler convolutions with a row and a column vector. Thus, I used the Sobel operator for my gradient approximations.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad \text{and} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

Figure 11a. Sobel operator. G_x represents a horizontal gradient image, while G_y represents a vertical gradient image.

$$G_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * ([-1 \quad 0 \quad 1] * A) \qquad \text{and} \qquad G_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * ([1 \quad 2 \quad 1] * A)$$

Figure 11b. Sobel operator separability. Demonstrating further separability for more efficient evaluation of the convolution.

Although in practice the Lucas-Kanade optical flow constraints are often violated in a webcam video sequence, the algorithm is somewhat robust against these violations because the flow is computed from gradients across multiple pixels instead of just one.

2.2.6 Decision Making

I experimented with each of these algorithms in order to determine their feasibility for incorporation into Scratch. All of my testing was done on a laptop running Windows 7 64-bit. The laptop had an Intel Core i7 CPU Q720 @ 1.60 GHz, 4 Gb of DDR2 memory, a NVIDIA Quadro FX 880M graphics card, and a solid-state hard drive.

For object recognition and object tracking, I used OpenCV and C++ for prototyping. OpenCV is a popular open source computer vision library that contains many useful image processing and computer vision functions. It was recognized that C++ applications might perform significantly faster than Flash ActionScript applications, thus the speed considerations were kept in mind when testing.

I was able to directly find implementations of SURF, k-means clustering, and BoW in OpenCV to test object recognition. The object recognition worked well in a very controlled environment. I experimented with creating a library of hand drawn symbols, and then comparing them against a video of one of the symbols. As expected, the algorithm was robust against rotation, scale, and lighting. However, the recognition was computationally expensive and took approximately five seconds to perform recognition on one frame with only a database of two symbols. Additionally, the recognition did not perform as well once the drawings of the symbols deviated slightly from those used to build the vocabulary tree.

Considering the potential use cases in Scratch, it seemed that current object recognition techniques would not be fast enough for real-time application with a webcam. Additionally, the limitations of the detection would seemingly limit the creativity of the user. One could easily imagine cases where a non-technical user might wonder why his hand-drawn circle did not match a similar but non-identical hand-drawn circle in the vocabulary tree. It would be difficult to explain the possibility of spurious matches being reported to a user that did not understand the algorithm. Thus, object recognition was omitted from the list of possible algorithms to implement in Scratch.

Object tracking was also omitted for similar reasons. It worked decently under very controlled environments. However, it also required seconds to finish computing a result for each frame, and would lose track of the object in noisy backgrounds. The design philosophy of Scratch requires that a block has an intuitive behavior even when used "improperly" instead of causing an error. The relative unreliability of the tracking would result in unintuitive behavior. As such, object tracking was also omitted from being implemented for Scratch.

For face tracking and optical flow, sample implementations were found in ActionScript. They were able to operate at a reasonable frame rate (requiring less than one frame time to compute a result). However, face tracking was still decided against because of it seemed to have limited application aside from being used as an x-y position. Scratch incorporates the most fundamental yet powerful functionality, and detecting the position of a face seemed to not provide much more than simply moving the mouse.

Optical flow was ultimately chosen as the algorithm to implement for Scratch. It could compute a result in approximately 6 ms per frame (relative to a webcam pulling 30 frames per second having a frame time of 33 ms) which was an acceptable speed. Additionally, it was quite accurate at following motions, and had some fair robustness against changes in brightness. It also seemed to have great potential for many creative and different uses, as was demonstrated in the user studies described later in the paper.

3. Implementing Optical Flow in Scratch

3.1 Introducing the Video Sensing block

Figure 12. The Video Sensing block.

The Video Sensing block was thus designed in order to reproduce the Lucas-Kanade algorithm for optical flow. The block is placed in the "Sensing" block category. Although all of the functionality is contained within one block type, the two drop down menus provide access to specific information.

The first drop down menu has two possible selections, "motion" or "direction." When the "motion" option is selected, the magnitude of computed motion is returned as a positive number proportional to the amount of motion. When the "direction" option is selected, the dominant direction of the motion is returned.

The second drop down menu also has two possible selections, "Stage" or "this sprite." These options allow the user to control the scale at which the optical flow is computed. If "Stage" is selected, then the motion detected in the entire frame size of the webcam feed is used to compute the magnitude and direction of optical flow.

To preserve some notion of scope, the "Stage" option is selectable when the Video Sensing block is used either at the Stage level or within a sprite. The other option is used to reflect the motion local to a specific sprite. Thus, the values returned by the block reflect only the optical flow sensed under the active pixels of the individual sprite. A sprite can only sense its own optical flow, and not the flow of other sprites. So, when a script using the Video Sensing block's "this sprite" option is copied to another sprite, then the meaning of "this sprite" switches to refer to that sprite.

3.2 Implementation Specifics

When a Video Sensing block is activated in a script, Scratch automatically initiates a Flash request to access the webcam, and the user is prompted with a dialog box to approve the request. Once this is accepted and the webcam begins sending valid frames to Scratch, the optical flow computation begins, and continues as long as valid frames continue to be received. If an invalid frame is received (the webcam

is stopped), the optical flow computation halts until valid frames are once again detected. In the event that the user has no webcam, no cycles will be spent on optical flow computation.

For Scratch, optical flow is computed at two scales: global (Stage-wide) and local (sprite-specific). Both rely on the same image pixel data. They differ only in that they use different regions of the frame. The webcam frames are always scaled to a 480x360 resolution to match the resolution of the Scratch stage. All of the pixels in the frame will be used for the global optical flow calculations, while only the active pixels of a sprite will be used for the local optical flow.

Because a webcam inherently has some level of sensor noise, a minimum threshold for motion was implemented to reject extremely small magnitudes to avoid reporting jumpy values. This threshold was set relatively low, understanding that each webcam has its own signal-to-noise ratio, and that users could define their own thresholds at the Scratch script level if necessary.

3.3 Design Considerations and Tradeoffs

Presentation to the User

I wanted the user to be able to quickly develop an intuition for using the Video Sensing block simply from experimenting with it briefly. However, the notion of optical flow was not trivial to convey concisely, so the wording and presentation had to be chosen carefully.

In the original prototype, the optical flow outputs were divided among four blocks: "global motion amount," "global motion direction," "local motion amount," and "local motion direction." The final version incorporated all of the functionality into one block for efficiency and ease of understanding. It was less intuitive to users not versed with optical flow to have separate blocks based on only one algorithm. Also, it would be easier for users to experiment with the different abilities of the algorithm if they could simply change their drop-down menu selections instead of dragging a new block for each change.

The word "video" was also added to help explain the usage of the block. It was difficult to convey the meaning of optical flow magnitude in one word, but ultimately "motion" was chosen, and its adjacency as a menu choice to "direction" was intended to highlight the distinction between the two.

The mathematical details of the algorithm were intended to be completely hidden from the user. However, it was intended that the user still be able to understand the general concepts of optical flow from observing how the Video Sensing block could be used.

It was also important to consider the notion of scale. Most sensor data in Scratch can be mapped into a predictable range such that a user can develop an intuition based on a number. However, for optical flow computation, it is not obvious how such a scale can be applied. For example, should the same amount of motion through a large sprite versus a small sprite map to the same number, or should it scale based on the size of the sprite? How do you define the maximum motion?

Ultimately, I decided not to implement a scale for optical flow for a few reasons. First, there is too much variation in the hardware involved. There are many varieties of webcams with different resolutions, frame rates, and signal-to-noise ratios. Therefore, it does not make sense to map fixed scale to an unpredictable range of sensor values. Additionally, having a fixed scale for each sprite would also carry a physical intuition with it. For example, a scale might confer the meaning that it is easier to "move" a smaller sprite than a larger sprite. However, this intuition might be undesirable, as in the case of displaying a large sprite of a feather and a small sprite of a stone. Finally, optical flow is more accurately computed when more pixels are used because noise effects are minimized. However, scaling the motion of a small sprite with few active pixels would also scale up the effects of noise, resulting in nonsensical values being reported.

Technical Decisions

When the Video Sensing block is activated, it automatically triggers the webcam to turn. A "turn video on" block also exists for the purposes of just activating the webcam independently of the Video Sensing block.

A new "set video transparency to x" block was created, motivated by the webcam usage of the Video Sensing block. Now, users that want to use the webcam but still display a background on the Stage can adjust the transparency level to suit their needs.

Through considering possible use cases of the Video Sensing block, an optimization of the algorithm to avoid redundant calculations arose. The spatial and temporal gradient information is stored globally for each frame whenever a Video Sensing block is being used. Therefore, when computations for overlapping regions would be needed, the gradient information would not need to be recomputed. These cases

included using global and local optical flow simultaneously, or using the optical flow of multiple sprites that might have overlapping areas. This solution will always restrict the computation to at most the amount necessary for one global scale optical flow computation per frame regardless of the number of times the Video Sensing block is used.

There are various discrete differential operators that can be used to estimate gradients in images. Their accuracy can be somewhat proportional to the amount of operations used. For example, using the Sobel operator to find the horizontal gradient of one frame would require K^2N^2 operations, where K is the width of the kernel, and N is the width of the image. Whereas a much simpler horizontal gradient estimator, such as multiplying a small local window of three pixels by [-1 0 1] requires just KN operations per frame. Accordingly in my experiments, it took approximately 18 ms to compute optical flow for one frame using the Sobel operator, and about 6 ms to compute using the simplest gradient estimation technique. The amount of accuracy difference seen was qualitatively negligible in this situation, and since the optical flow computations were not meant to be very precise, it was worth saving computation time in trade for a slightly less accurate estimate.

4. Demonstrative Example Projects

To demonstrate the Video Sensing block, I present several sample Scratch projects that make use of its capabilities.

4.1 Miniature Vibraphone

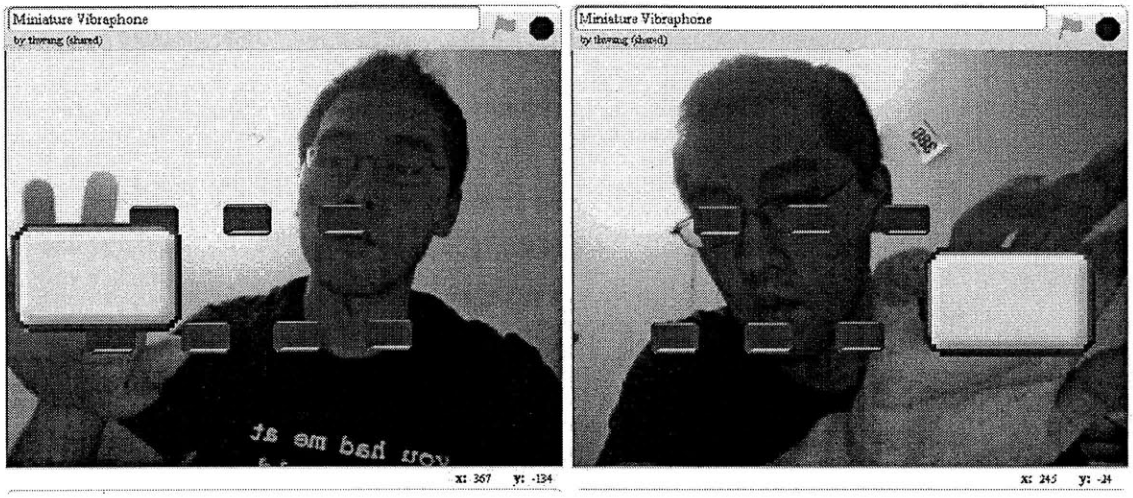

Figure 13. Miniature Vibraphone project. Showing that the mallet can be controlled by moving my hand underneath it to change brightness gradients in the direction of my hand motions.

In the Miniature Vibraphone project, the user is able to control the movement of a white "mallet" sprite using its local optical flow, and as the mallet comes in contact with the purple bars, a vibraphone tone is generated. Thus, a simple musical instrument can be played.

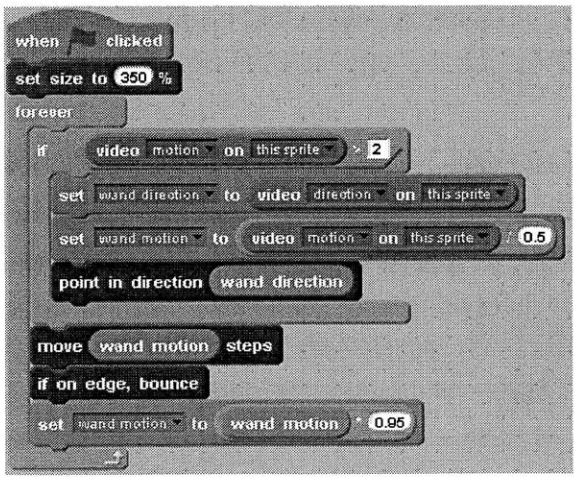

Figure 14. Wand script.

The script for controlling the wand is displayed in the above figure. A forever block is used in conjunction with an if block. If the video motion ever exceeds a certain small threshold (to filter out noise), then the video direction and motion can be used to define the speed and orientation of the wand. The wand is then moved according to the computed optical flow, and its speed decays to 95% at each loop iteration to simulate a more natural feeling of speed decay.

Two main concepts are introduced through this example. First, optical flow can be continuously sensed and computed once per frame. Secondly, a sprite can be moved in a way that is proportional to the amount of motion as well as the direction of the motion computed beneath its active pixels.

4.2 Play with Balloons

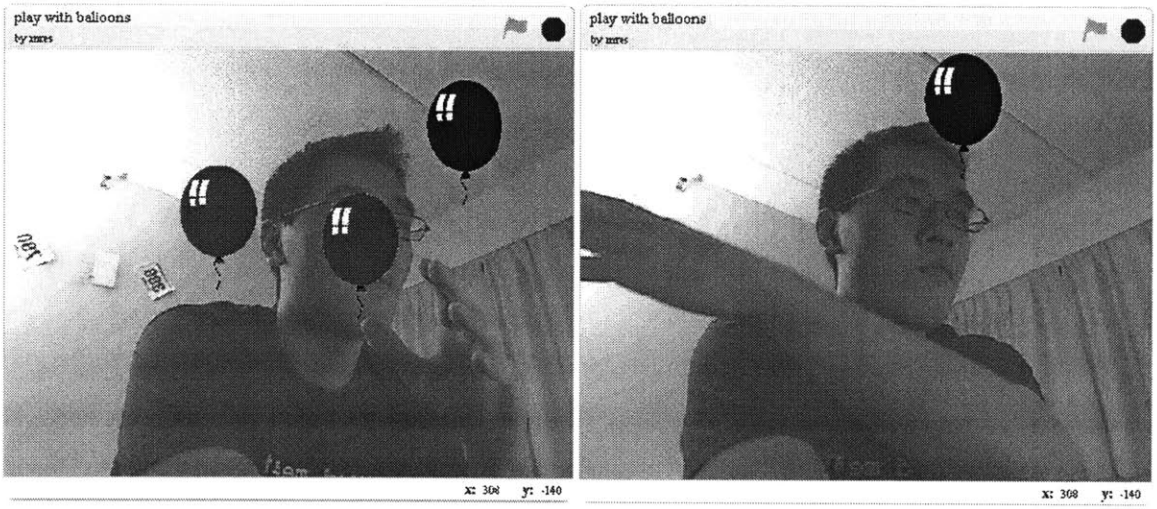

Figure 15. Play with Balloons project. Swiping my hand across the red balloons pops them and also causes the blue balloon to move in the direction of my motion.

In this project, the user plays with two types of balloons. The red balloons can be popped, while the blue balloon can be moved. The scripts for each balloon type are presented below.

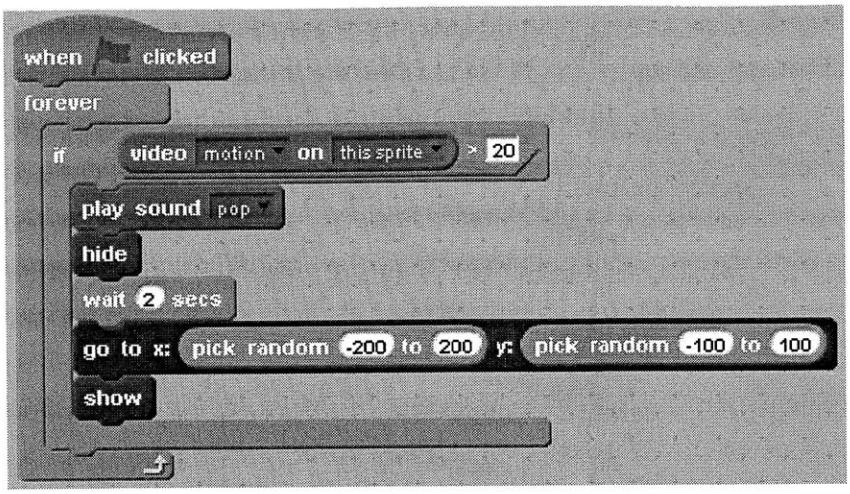

Figure 16. Red balloon popping script.

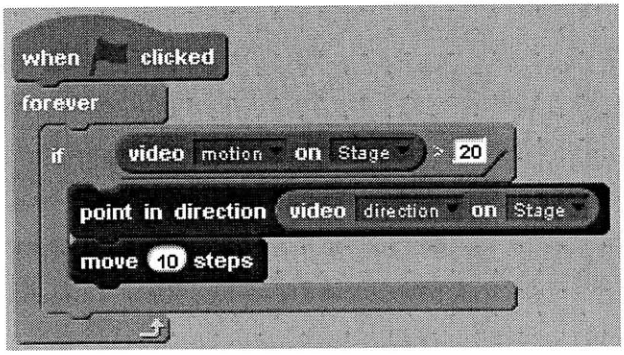

Figure 17. Blue balloon motion control script.

The red balloons float randomly until the local optical flow magnitude exceeds a certain threshold, at which point they pop, and reappear after a short delay. In contrast, the blue balloon observes the global optical flow of the entire stage, and if the motion magnitude exceeds a threshold, it moves a fixed amount of steps.

The concepts displayed in this project are the notion of a Boolean state being that can be toggled by exceeding a certain amount of local motion magnitude without regards to direction, and also the concept of using the entire Stage's optical flow to control the motion of a sprite. This motion control scheme differs from that of the vibraphone project's wand sprite in that although the direction is used, the amount of motion here is fixed and not proportional to the magnitude of the motion. This logic is useful when the user prefers a smooth rather than sensitive response to the optical flow.

4.3 Webcam Pong

Figure 18. Webcam Pong project. Demonstrating multiplayer capability.

The Webcam Pong project is an adaptation of the classic game of Pong to use optical flow. Each paddle sprite is controlled by its local optical flow. At the beginning of each round, the soccer ball takes a semi-random velocity and points are scored when a player bounces the soccer ball past the opponent's paddle to the opposing wall.

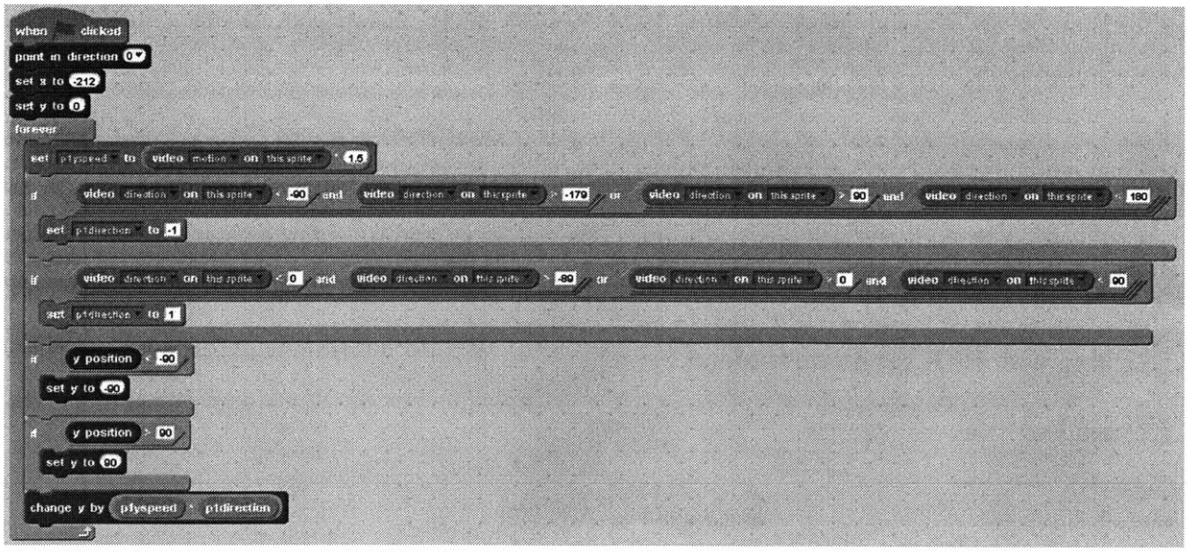

Figure 19. Pong paddle script. The [-180,180] range of video direction is split in half to correspond to the two possible directions of movement.

Each paddle is controlled by its local optical flow. The paddle's motion is constrained to only run in the vertical direction. It will move upwards if the direction of the computed optical flow has an upward component, and it will move downwards if the converse is true.

This project highlights that multiple sprites can be controlled simultaneously by optical flow, and also encourages a new form of interaction: multiplayer. Since optical flow only relies on changes in brightness, there is no restriction on how those changes are achieved. My hope is that the Video Sensing block will accordingly create new types of projects that encourage multiple users to participate at once.

5. User Studies

After I had finished implementing the Video Sensing block, I wanted to see how well it would be received by Scratch users. I was curious to discover how close the intuitions developed from playing with the block would reflect the optical flow algorithm implemented beneath the surface. Thus, I decided to perform some studies with Scratch users.

It made the most sense to test the new block type with users who were already quite familiar with Scratch. These users would more easily be able to focus on understanding the Video Sensing block rather than be distracted by learning the Scratch environment. Also, it would be educational to test with users of varied ages. The math required to derive the optical flow equations is taught at the college level, but I believed that the ability to develop an intuition about the Video Sensing block's functionality would be present in much younger users.

Accordingly, I introduced the Video Sensing block to two different groups. The first group consisted of the LLK graduate students. The second group was more varied, consisting of about 20 Scratch 2.0 alpha testers. The following sections describe the test setup and results for each group.

5.1 Testing with LLK graduate students

5.1.1 Test Logistics

The LLK graduate students were already aware that I was working on optical flow prior to the actual test. I had given them a brief technical presentation on optical flow as well, thus giving them some familiarity with the concepts involved. However, they did not have an in-depth introduction to the specific block

implementation until the time of test. Additionally, at the time of the test, the optical flow functionality was still split up into four distinct blocks, and no video transparency block was available.

At the beginning of the hour-long meeting, I refreshed the LLK graduate students' memories on the idea of optical flow. Then, I gave them about 40 minutes to create their own projects using the optical flow blocks and share amongst themselves. We all sat at the same table which facilitated communication. To help demonstrate how to use the blocks, I also gave them access to a set of demonstrative sample projects that I had created prior to the session. Thus, they had the option to "remix" or build upon my sample projects instead of starting from scratch. The three sample projects I provided were earlier versions of the sample projects described in section 4.

5.1.2 Observations and Results

It was instructive to observe the design process of the other students. All of them quickly understood the function of the blocks. Once they felt comfortable with the functionality, their projects quickly diverged in content. I will describe a sampling of the projects below.

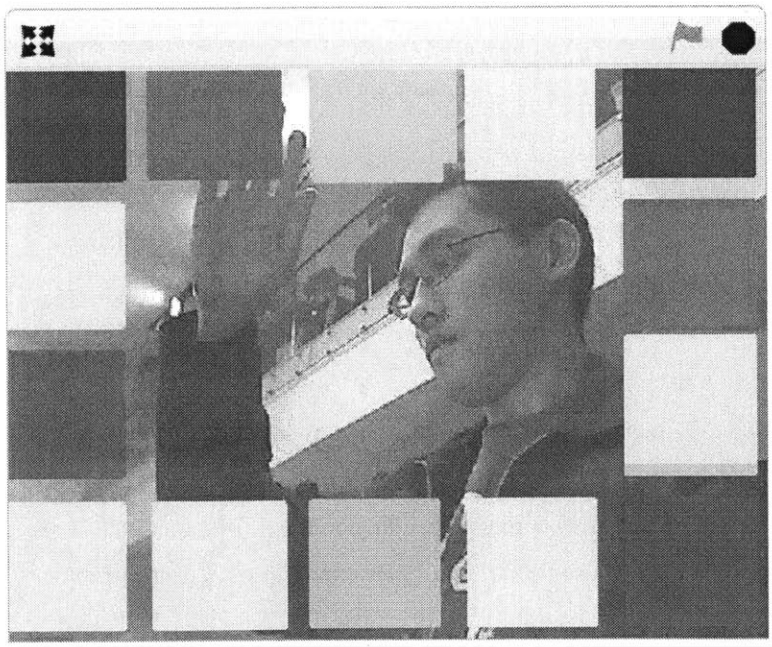

Figure 20. Colorful Tiles! project. Equally entertaining to experiment with a completely opaque background (no video displayed).

First, there was the "Colorful Tiles!" project. A solid color square was tiled along a rectangular grid on the stage. When the local motion magnitude of a tile exceeded a threshold value, the tile would change color. The net effect was that the user could then move around the screen and observe the tiles changing color in a pattern that followed his/her motion. Not only was there a nice artistic effect at play, but also the notion of a motion sensor.

From watching the creation of this project, I learned that defining a motion magnitude threshold should be done manually by each user for his or her specific environment and hardware. Unpredictable variations in the webcam feed would prevent me from defining one acceptable motion threshold that could be used universally. I also learned that optical flow could be used to create artistic projects in Scratch.

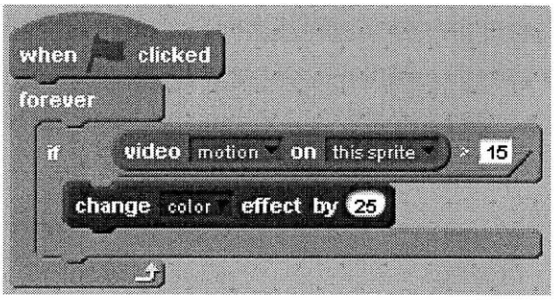

Figure 21. Tile script.

Another interesting project was called "totoro." This project made use of both global and local optical flow. The premise was that global motion greater than a threshold would cause the tree to grow in size, while global motion less than that threshold would cause the tree to shrink. The user would accordingly have to continue creating motion to grow the tree and keep it from shrinking. A feedback loop is created, where the user input is constantly adjusted in response to visual feedback. Additionally, a Totoro character sprite would respond to its computed local motion and quickly move away from strong motions.

Through observing this project, I learned that optical flow could be thought about not only for small bursts of intentional gestures, but that it could also be used continuously (and indefinitely) to influence metrics over extended periods of time.

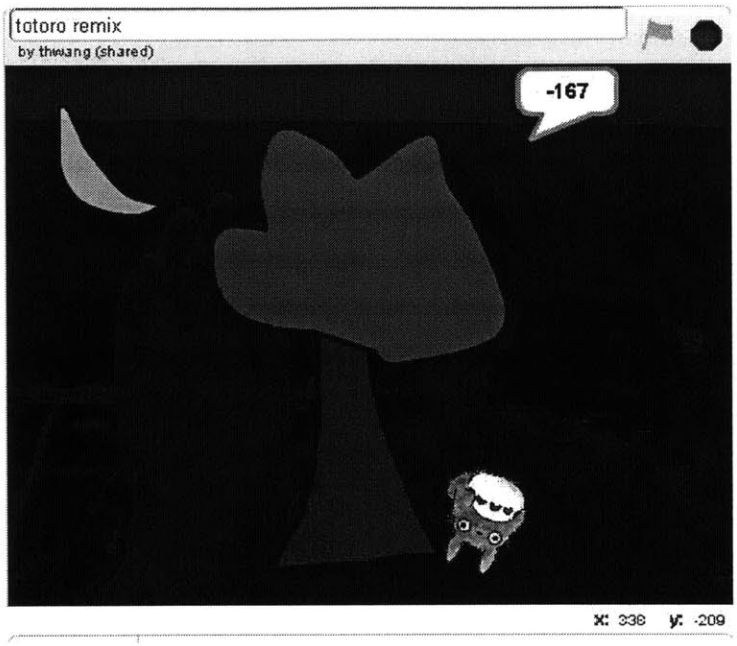

Figure 22. totoro project.

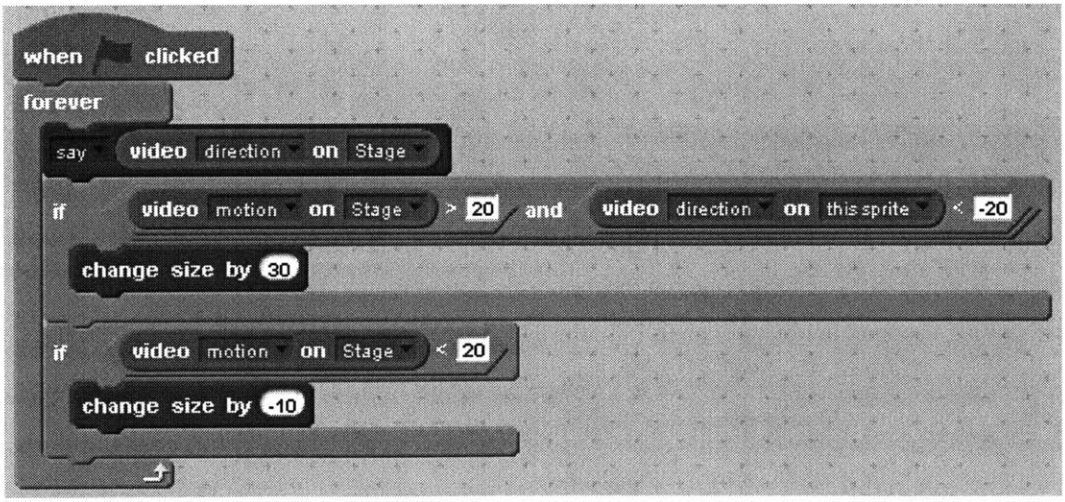

Figure 23. Tree script. The motion exceeding a threshold in a specific range of directions causes the tree to grow larger. If the motion dips below the threshold, the tree shrinks, albeit at a slower rate than it grows.

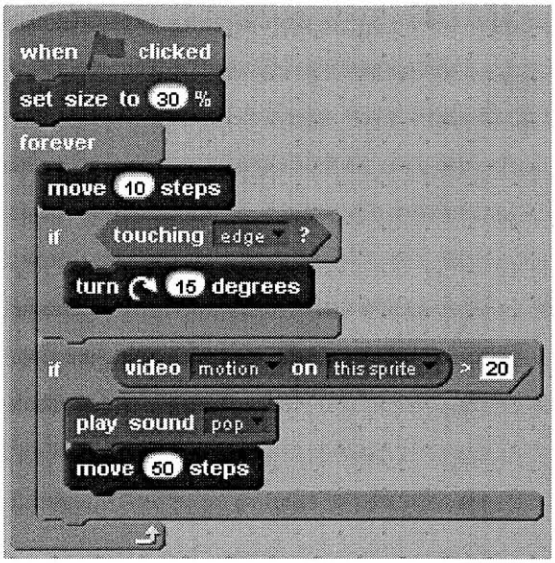

Figure 24. Totoro script.

A final example from this user study was the "motion painter" project. This project used a small tennis ball sprite as a paintbrush that the user could control with optical flow. The ball would leave a pen trail by default as it moved around the stage. However, the user could also toggle a button that would lift the ball such that it could move without leaving a trail. This button was toggled again by exceeding a certain local optical flow magnitude threshold.

The "motion painter" project revealed a few interesting things about the optical flow algorithm. First, it was pointed out that motion was harder to detect for small sprites such as the ball. This highlighted one of the inherent shortcomings of the optical flow implementation. Additionally, creative ways of circumventing the shortcomings were revealed in the user's method of interaction with the project. Instead of just moving his hand behind the tennis ball, the user instead created a ring with the fingers of his two hands that surrounded the tennis ball and used this hand positioning to move the ball. The ring shape around the ball acted as a sort of buffer that could help prevent the algorithm from losing track of the ball's local optical flow despite the sensitivity issues.

Figure 25. motion painter project. The green button has been toggled a few times resulting in distinct pen trails.

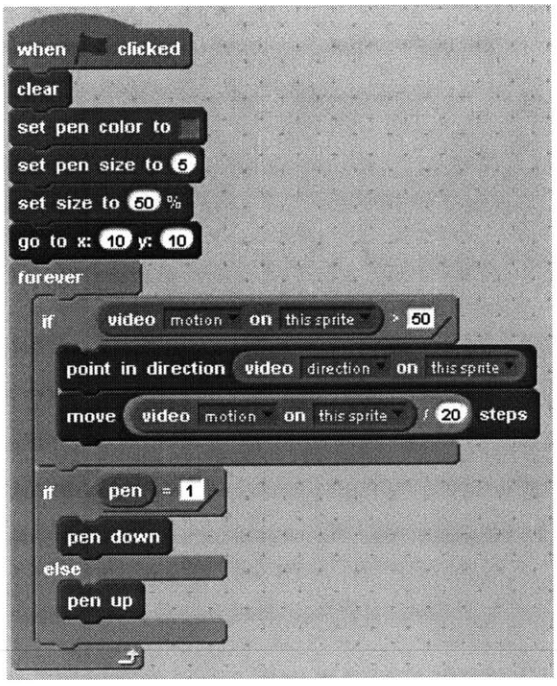

Figure 26. Ball/pen control script.

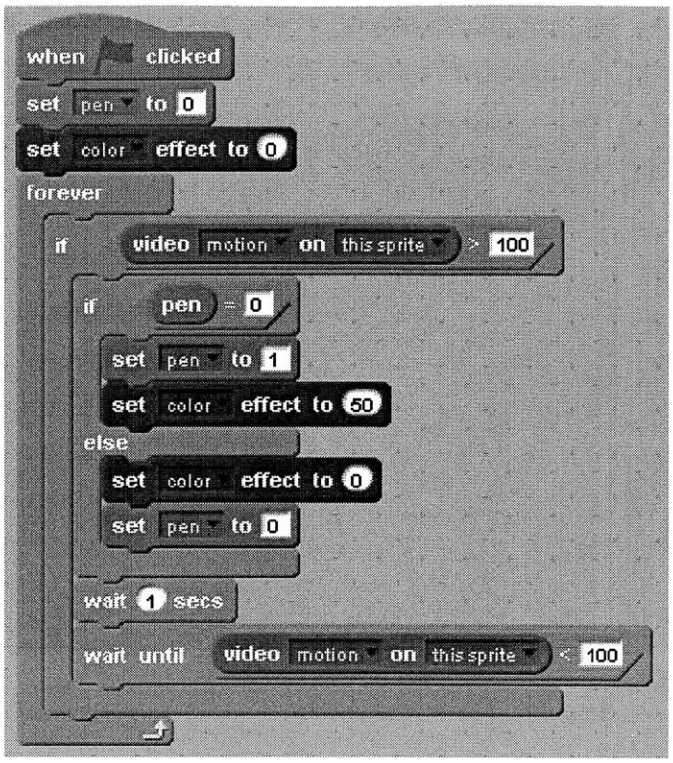

Figure 27. Pen toggling button script.

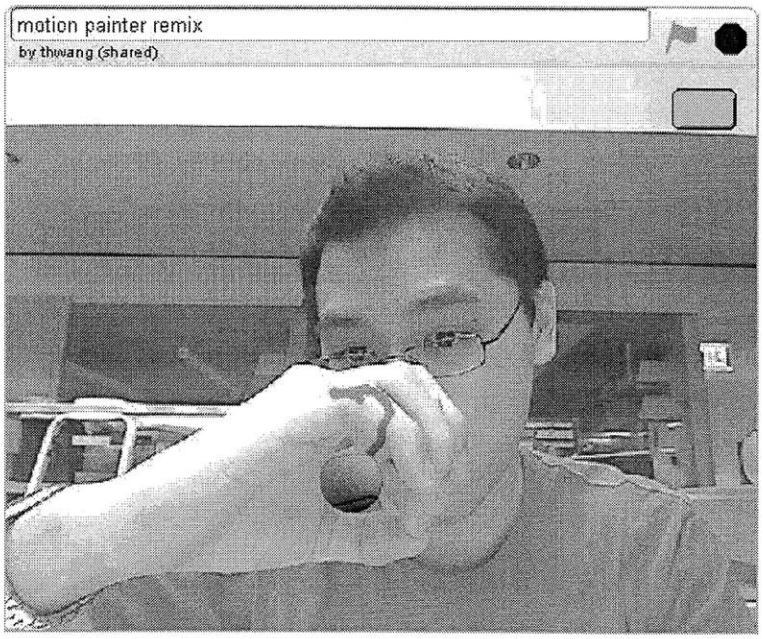

Figure 28. A technique for controlling local optical flow of a sprite. Forming a ring around the ball helps keep it from straying.

5.2 Testing with Scratch alpha testers

5.2.1 Test Logistics

After considering the results of the testing with the LLK graduate students, the optical flow blocks were condensed into the Video Sensing block, and video transparency was made available. Then, these updates were released on our test server to a group of Scratch 2.0 alpha testers to try. They were not given a background explanation of optical flow, nor were they even informed that the Video Sensing block was computing optical flow. Instead, they were just provided this forum post:

"For this week's Topic of the Week, we're hoping that you'll try out a new feature of Scratch 2.0 -- Video Sensing. You can now make Scratch projects in which people use body movements to interact with sprites, in the spirit of Microsoft Kinect. For example, you can make a volleyball game where people hit the (virtual) ball with their hands.

You'll need a webcam on your computer to use Video Sensing. We've added a new block, in the Sensing category, that allows you to detect the amount and direction of motion in the video from the camera. You can detect the motion underneath a particular sprite, or over the entire stage. There are also new blocks in the Looks category of the Stage, enabling you to turn the video on and off, and to control the transparency of the video.

For some examples of projects using the new Video Sensing feature, see this gallery of projects.

We encourage you to try out these projects -- and create your own projects using the new Video Sensing blocks. We look forward to hearing your comments and suggestions, either in this forum or by sending email to scratch2-test@media.mit.edu

Mitch Resnick for the MIT Scratch Team"

I was curious to see what intuitions would be developed by the alpha testers because they were not informed that the Video Sensing block was computing optical flow. I still believed that they would be able to grasp how to use the block just as quickly. Additionally, the alpha testers consisted of users with a wide variety of educational backgrounds. On the younger end of the spectrum were children that have been active contributors to the Scratch community, while on the other end were educators and teachers that have supported use of Scratch in the classroom.

5.2.2 Observations and Results

Reading the forum posts by the alpha testers allowed me to follow their thought processes as they tried to understand what was happening inside of the Video Sensing blocks. Within a day or two of the feature being announced, some misconceptions were formed on what was happening. For example, one user

thought that movement was only being quantified in the horizontal direction. Another user thought that only color changes were being measured.

However, given a few more days to play with the software, gradually the users figured out almost exactly what was going on. One of the more advanced users was able to guess that the motion was being calculated as the integral of motion vectors in a specified area. Although he formed an analogy to acceleration, which is not entirely accurate (an analogy to velocity would be more appropriate), he was still able to create some compelling projects that made use of the block as shown below.

Figure 29. windchime project. The notes bounce off of each other in the horizontal direction, creating an interesting effect when many notes are introduced to the system.

The "windchime" project uses the local optical flow magnitude on a series of notes to control their motion. Instead of using the magnitude as a velocity as has been seen in previous projects, it uses the flow as an acceleration. In defining a physical system like a set of wind chimes that can collide, basic concepts of Newtonian physics are present.

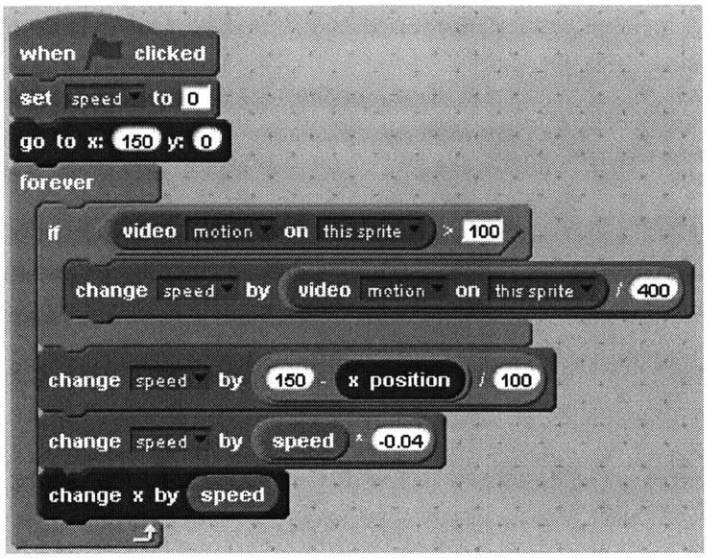

Figure 30. Script of a chime/note. Here we see speed being changed rather than position being changed.

Another use for the Video Sensing block surfaced in the "Chatroom motion detecting" project. The goal of this project was to automatically send chat room messages indicating whether a chat participant is away from the computer. The program assumes that a person will generate a certain amount of motion while they are in front of a computer, and when they leave, the motion will fall below a threshold. Thus, if the user is gone for a set idle time, the program could automatically report that they are idle. Once motion is detected again (the person returning), the status could be reset to available.

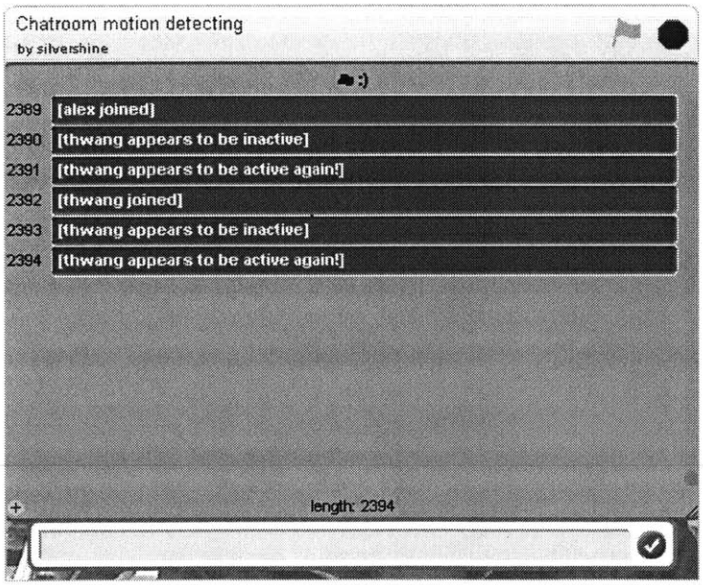

Figure 31. Chatroom motion detecting project.

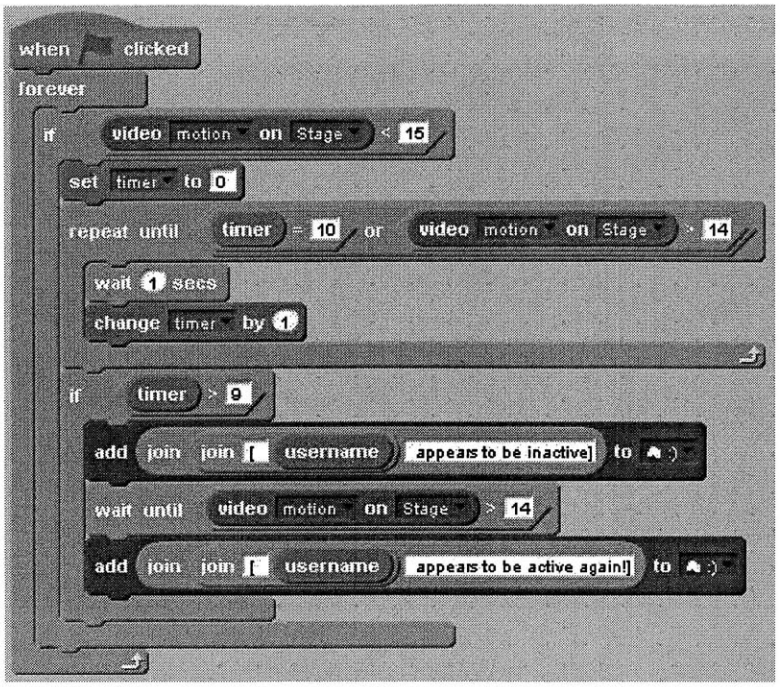

Figure 32. Script for state determination. This script resides on the Stage level.

While making this project, the alpha tester became very aware of the behavior of an imperfect webcam sensor. He was able to deduce that there was a degree of sensor noise that would always be present, and that sometimes there would be noise spikes either at random or due to environmental changes.

6. Analysis

6.1 Comments on User Studies

Overall, I found the user studies to be successful. There were different rates of user comprehension of the Video Sensing block. Some users were clearly able to deduce more meaning as to the specifics of the optical flow algorithm. However, eventually everyone was able to understand how to use the block through experimentation. I also received feedback that the sample projects were extremely helpful for helping them get started. As sharing and remixing projects is the nature of Scratch to begin with, I see no issue with this avenue of learning.

A wide variety of projects were made during both studies. The alpha testers had approximately one week to test the Video Sensing block, and with this time were able to demonstrate some compelling ideas and new uses for optical flow. The LLK graduate students had a relatively short period of time to create, and

yet they still produced engaging and entertaining projects. This result makes me believe that the Video Sensing block has a low barrier to entry, yet significant depth for a user with motivation and time.

The overall enthusiasm over the new Video Sensing block was encouraging. Giving kids the opportunity to work on projects that they find personally meaningful is one of the best ways to keep them motivated and learning [17].

6.2 Potential Improvements

There are a few cases where the optical flow implementation breaks down, mostly related to situations where the constraints of Lucas-Kanade optical flow are violated. The most significant case is when extremely fast motions are not detected by the algorithm. A typical webcam can capture about 30 or 60 frames per second which may not be fast enough to smoothly capture very fast motions. Lucas-Kanade optical flow can only follow motion when the distance that a moving object has traveled between frames is approximately less than or equal to the object's width. Thus, during user testing, situations did arise where the user would move their hand very quickly, but the motion would not be detected. Also, users also reported that sometimes their repeated motions yielded different sensor readings. This effect may have been caused by either user input error, or by the fact that different frames of the motion were captured by the webcam.

The accepted computer vision solution to these problems is to compute the optical flow at different scales. This means that instead of just computing one scale of optical flow (as currently implemented), additional scales are computed for each frame, and optical flow is estimated at each scale. Once optical flow is computed for each scale, a weighted least squares analysis can be used to solve this system to find the most likely optical flow estimate.

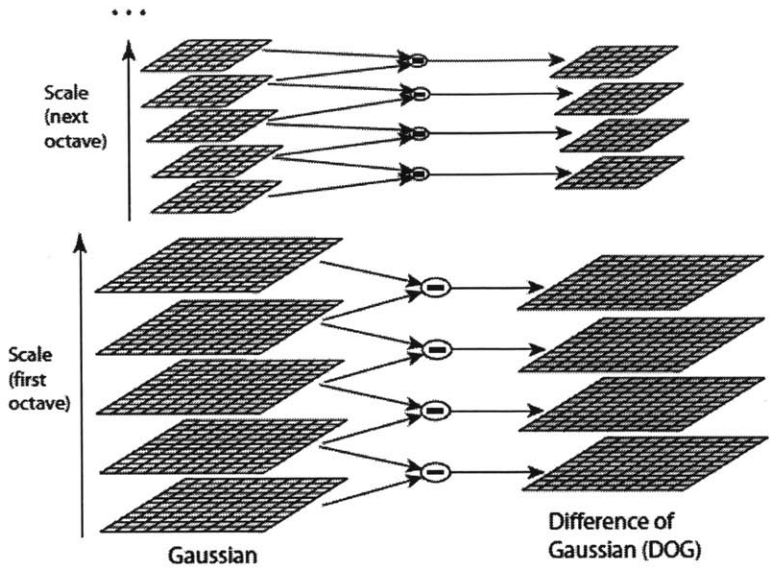

Figure 33. Multiple scales of an image. This image displays the formation of image scales by cascading convolutions with a Gaussian kernel.

To create the other scales, typically one applies a Gaussian blur to a frame, and cascades the blurring until the desired number of scales are computed. Intuitively, as the image becomes blurrier, larger features become more significant. Thus, moving a hand quickly across the screen in the original scale might violate the smoothness of motion constraint, but at a very large scale (very blurry), this motion may not be as large relative to the size of the hand at that scale. The tradeoff for implementing multi-scale optical flow is at least a linear increase in the amount of computation required.

Another area of potential improvement could be more accurate methods of estimating derivatives. The current implementation uses a very simple gradient estimator, and did not show noticeable accuracy improvement when using the Sobel operator. However, there are many other gradient estimators, and it may be possible to improve the accuracy by using a more optimal estimator. Again, the tradeoff for using a differential kernel with larger support would be greater computational complexity in estimating derivatives.

Related to the idea of better derivative estimates is the handling of the special case of small sprites. Because small sprites don't have as much area over which to estimate optical flow, they have a smaller range of possible magnitudes in practice. That is to say that a very fast motion across a very small sprite may still result in a smaller optical flow magnitude than a slow motion by a large object across a large sprite. Perhaps some estimates comparing the relative magnitudes of optical flow inside and immediately

outside in the neighborhood of the sprite could be used to help normalize the output of the Video Sensing block to something more intuitive, even if it doesn't faithfully reproduce the optical flow algorithm's results. After all, the typical user desires to have a block that responds intuitively to motion, not a block that precisely computes optical flow.

The notion of mapping optical flow magnitude itself to a fixed scale could be addressed by a similar method. Such a wide range of hardware variation in webcams and computers exists that it is not obvious how one could ever assign the flow to a scale of say 0 to 100. However, one could imagine some potential in the idea of using relative metrics such as comparing the amount of local motion to the amount of global motion. Also, perhaps having some amount of stored history of optical flow, or a calibration step could be used as valuable information to map the motion to a fixed scale.

Finally, further exploration could be done into the intuitiveness of setting the noise threshold level proportionally to the size of the sprite. Perhaps a signal-to-noise ratio could be estimated for the entire image, then applied to a smaller region to approximate an appropriate threshold.

References

[1] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. Communications of the ACM, vol. 52, no. 11, pp. 60-67 (Nov. 2009).

[2] Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. ACM Transactions on Computing Education (TOCE), vol. 10, no. 4.

[3] Rosenbaum, E., Silver, J. Programming with Color. URL http://web.media.mit.edu/~ericr/color.php

[4] Radu, I. AR SPOT: An Augmented-Reality Programming Environment for Children. URL http://ael.gatech.edu/lab/research/arspot/

[5] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L. (2008). SURF: Speeded Up Robust Features, Computer Vision and Image Understanding.

[6] Chiu, K. (2011). Vision on Tap: An Online Computer Toolkit.

[7] Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., Nordlund. P. (2001). Particle Filters for Positioning, Navigation and Tracking. IEEE Transactions on Signal Processing.

[8] Isard, M., Blake, A.(1998). CONDENSATION – Conditional Density Propagation For Visual Tracking. University of Oxford.

[9] Sun, L., Liu, G. (2011). Object Tracking Based on Combination of Local Description and Global Representation, IEEE Transactions on Circuits and System for Video Technology, Vol. 21, No. 4.

[10] Ta, D., Chen, W., Gelfand, N., Pulli, K. (2009). SURFTrac: Efficient Tracking and Continuous Object Recognition using Local Feature Descriptors, CVPR.

[11] Wang, T., Gu, I., Khan, Z., Shi, P. (2009). Adaptive Particle Filters for Visual Object Tracking Using Joint PCA Appearance Model and Consensus Point Correspondences.

[12] Viola, P., Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features, CVPR.

[13]Weber, J., Malik, J. (1995). Robust Computation of Optical Flow in a Multi-Scale Differential Framework. International Journal of Computer Vision, 14, 67-81.

[14] Lucas, B., Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. Proceedings of Imaging Understanding Workshop, pp. 121-130.

[15] Lowe, D. (2004). Distinctive Image Features for Scale-Invariant Keypoints. International Journal of Computer Vision.

[16] Horn, B. K. P. (1986). Robot Vision. MIT Press.

[17] Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. Basic Books. New York.

[18] Resnick, M. (2003). Playful Learning and Creative Societies. Education Update, vol. VIII, no. 6, February 2003.

[19] Resnick, M. (1996). Distributed Constructionism. Proceedings of the International Conference of the Learning Sciences, Northwestern University.