Trapped

Olivier Leprince

Kosta Kaplanis

Mohannad El-Jayousi

TRAPPED
Real-Time Video Processing Tool
Final Project

Presented to
Professor Sha Xin Wei

Olivier Leprince :: 5521297
Kosta Kaplanis :: 4873475
Mohannad El-Jayousi :: 5170214

Concordia University
COMP471/2F
December 2006

TABLE OF CONTENTS

Section Page

1.0 INTRODUCTION 1

2.0 THE TEAM 2
- 2.1 Members
- 2.2 Project Goals
- 2.3 Roles and Contribution

3.0 EVOLUTION OF A CONCEPT 4
- 3.1 Initial Concept: Infinity Cubed
- 3.2 Trapped: The Rehash
- 3.3 Technical Interest
- 3.4 Presentation Installation

4.0 DESIGN AND IMPLEMENTATION 8
- 4.1 Background Subtraction *(by Kosta Kaplanis)*
- 4.2 Edge Detection: Aggressive *(by Mohannad El-Jayousi)*
- 4.3 Finding Key Point Coordinates *(by Olivier Leprince)*
- 4.4 NURBS: Web Creation *(by Olivier Leprince)*
- 4.5 Trapped Patch
- 4.6 Constraints and Challenges

5.0 MILESTONES AND DELIVERABLES 28
- 5.1 Project Lifespan/ Milestones
- 5.2 Deliverables

6.0 CONCLUSION 30

REFERENCES 31

LIST OF FIGURES 32

APPENDIX I – INSTALLATION 33

APPENDIX II – TRAPPED PATCH 34

APPENDIX III – BACKGROUND PATCH 35

APPENDIX IV – EDGE DETECTION (AGGRESSIVE) PATCH 36

APPENDIX V – FINDCOORDS PATCH 37

APPENDIX VI – NURBSCONTROL PATCH 38

1.0 INTRODUCTION

Trapped is a real-time video analysis tool created using MaxMSP and Jitter. This report thoroughly documents all relevant information regarding the project. This thorough documentation includes project goals, team members and their roles, the evolution and technical interest of the concept, a detailed description of the design, implementation and mathematics of the all the elements that make up the tool, challenges and constraints, and finally a project timeline with milestones and deliverables.

2.0 THE TEAM

2.1 Members

Team Trapped is made up of three engineering students: Olivier Leprince in his second year of Computer Engineering, Kosta Kaplanis in his final year of Software Engineering, and Mohannad El-Jayousi in his final year of Software Engineering. Prior to taking this course, none of the three team members have had any background in video analysis/processing, graphic design, OpenGL, MaxMSP or Jitter.

2.2 Project Goals

Despite the lack of experience in the field, the team was ambitious from the start. The goal was to gain skills over the semester. These skills consist of but are not limited to familiarizing ourselves with MaxMSP/Jitter (from the tutorials) and video analysis/processing techniques (from the lectures) and to collectively apply them to create a tool that mainly concerned itself with analyzing video. Though such a tool strictly made for video analysis would satisfy the engineering/mathematical appetite of the team, it is by itself a bit dry. For this reason a minor creative/artistic side was to have a presence in the project to spark interest among the general public. This creative side would also fulfill the team's interest in creating 3D objects, either OpenGL or otherwise.

In short, the overall project goal was to create an end product that analysed video in a systematic/mathematical manner, but had a superficial artistic output/concept for the benefit of a viewing audience.

2.3 Roles and Contribution

Team Trapped worked tightly together, with an equal amount of effort and contribution from every team member. Each member had a hand in every aspect of the whole project, though the roles and contributions were separated in the following manner:

- ***Olivier Leprince:*** Responsible for the FindCoords and NurbsControl patches. Found various mathematical solutions to video analysis issues.
- ***Kosta Kaplanis:*** Responsible for BackGround patch, and its discarded predecessor. Took charge of research and testing/troubleshooting activities.
- ***Mohannad El-Jayousi:*** Responsible for the discarded "aggressive" Edge Detection patch. Web site and video montage designer.

3.0 EVOLUTION OF A CONCEPT

The final product bases itself on an idea that transformed dramatically over the course of the project lifespan, though from start to finish, the basic concept maintained a certain integrity and direction. This evolution is documented in this section of the report.

3.1 Initial Concept: Infinity Cubed

From the very first meeting when the team was created, the basic foundation of the concept was agreed upon. The basis of the concept consisted of two ideas that have since appeared in every stage of the evolution of the project. The first idea was to analyze the video, and from this analysis, retrieve empirical data. This dealt with the mathematical nature of video analysis. The second idea was to take the retrieved data and use it to drive a graphical element in the output.

With the basis of the project concept established, the team then began elaborating on these two ideas. Unfortunately, at the time, the team was unfamiliar with the possibilities that MaxMSP, Jitter and the Computer Vision (cv.jit) library provided. Due to this unfamiliarity and the open-endedness of the project requirements, the initial project proposal was relatively simple and non-ambitious. The output was to be entirely in OpenGL: a cube floating in open space, taking for its trajectory the character 'alpha' (the symbol for infinity: ∞). The real-time video coming in through the camera would be used strictly to feed numerical parameters to the graphical output. The cube was to leave a trace behind it to represent its previous location, though this trace would only appear

momentarily. The 3D cube would have the ability to change its appearance (in parallel with the input's RGB average), its velocity (in parallel with the amount of movement from the input), and its rotational speed (depending on sound coming in through a mic). The general idea is illustrated below in Figure 3.1.1.

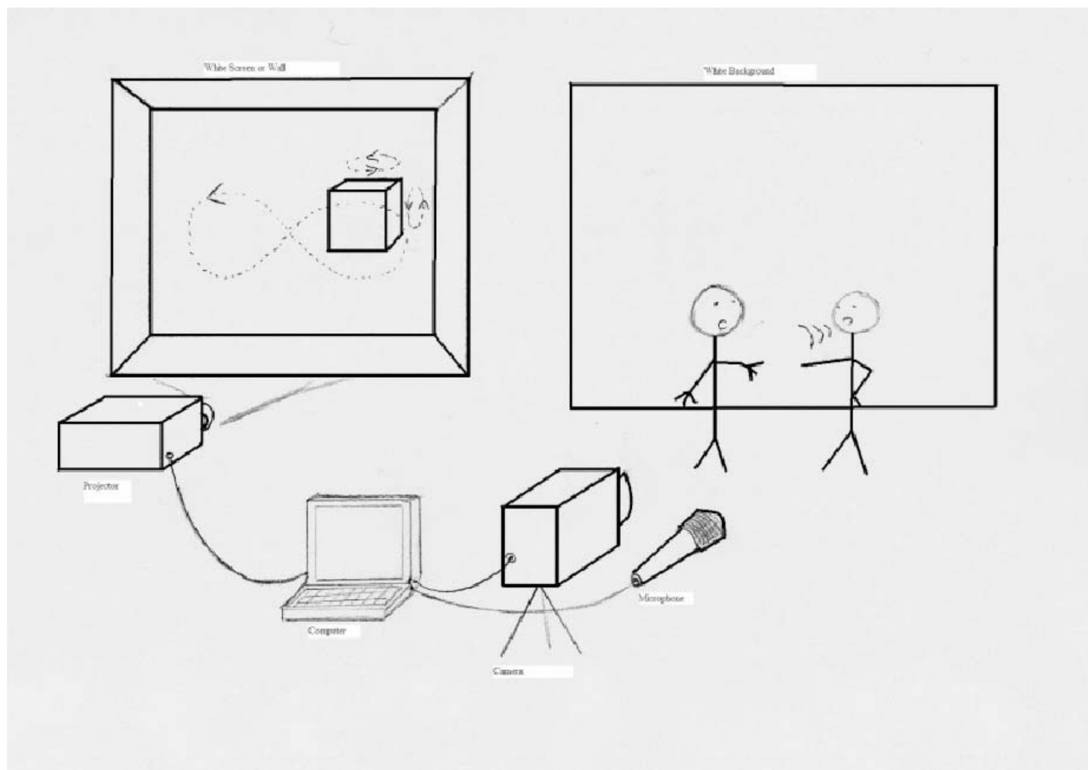


Figure 3.1.1 Initial proposal illustration depicting envisioned output/installation

Once this project proposal was presented by the team, Professor Sha Xin Wei signalled his concern about the relatively easy nature of the project: the video analysis was simplistic, and could have been accomplished using out-of-the-box objects from Jitter and the Computer Vision library. The team then applied several changes to the Infinity Cubed concept. This did not alleviate the plainness of the concept. Ultimately, the team decided to scrap everything except the two fundamental ideas that the project was based on and move in a different direction.

3.2 Trapped: The Rehash

The scrapped concept was rehashed to the following one: The new tool would have a more involved video analysis aspect, and would incorporate the video input in the output. Primarily, it would have a subject stand in front of the camera, and have his/her image projected with a (spider) web enveloping it. As the subject moves or extends his/her limbs, the web follows and stretches accordingly, giving the illusion that the subject is trapped in the web, hence the project name. See Figure 3.2.1 below for an illustration of the concept.

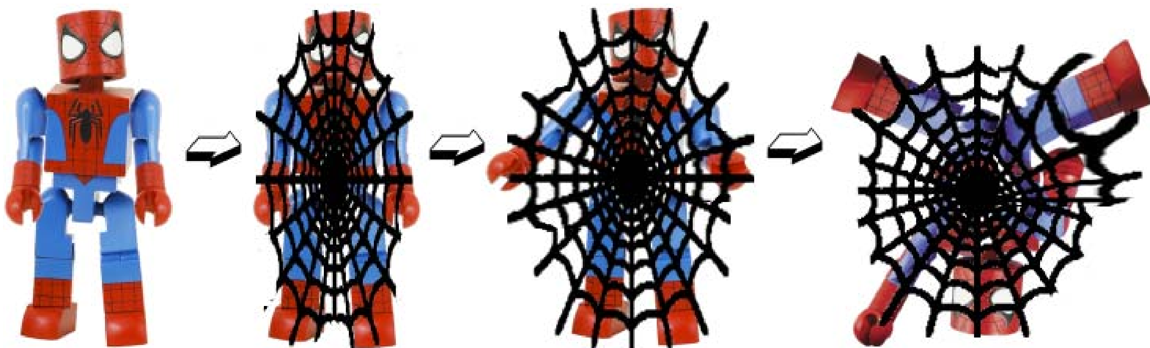


Figure 3.2.1 Spider web concept depiction

This concept was a far cry from Infinity Cubed, but retained the ideas of analyzing the video and utilizing the retrieved values to manipulate a graphical object. With a new concept hatched up, it was time to have the idea materialize into a tangible piece of work. Team Trapped was then ready to move on to the next stages of the project's development: design and implementation.

3.3 Technical Interest

Many interests were involved in choosing this project. First off, applying different video processing techniques such as background subtraction and edge detection to obtain different effects on video was obviously one of the main interests. The actual application of these techniques was a great challenge, and much learning and research was involved in applying these techniques successfully. Next, extending our knowledge of Jitter was another benefit of this project. Contrary to what was taught in tutorials for creating three-dimensional objects, the team opted for a different approach in accomplishing this task. The team was lucky enough to explore a technique not seen in class, the Non-Uniform Rational B-Splines (NURBS), a mathematical model commonly used in computer graphics for generating and representing curves and surfaces. It was a great challenge in using this model in the creation of a web which is able to interact with real-time movement through a live video feed. Finally, this project allowed for three individuals to apply their research and knowledge with a strong team spirit to generate an impressive working project that could be enjoyed by the general public.

3.4 Presentation Installation

For the Trapped tool to be demonstrated as it was for the live presentation, an installation like the one illustrated in Appendix I is required. Resources needed for such a setup are a homogeneous background, an iSight, a Mac computer with MaxMSP/Jitter, a projector, and a screen.

4.0 DESIGN AND IMPLEMENTATION

For the concept of Trapped to come to life, a certain process is needed to take place that entails several real-time video analysis and processing techniques, as well as the creation of a 3D object. This multi-step process includes background subtraction, grey scaling, edge detection, web creation, web manipulation, and superimposition. The process is illustrated in Figure 4.0.1 below, and elaborately explained in this section.

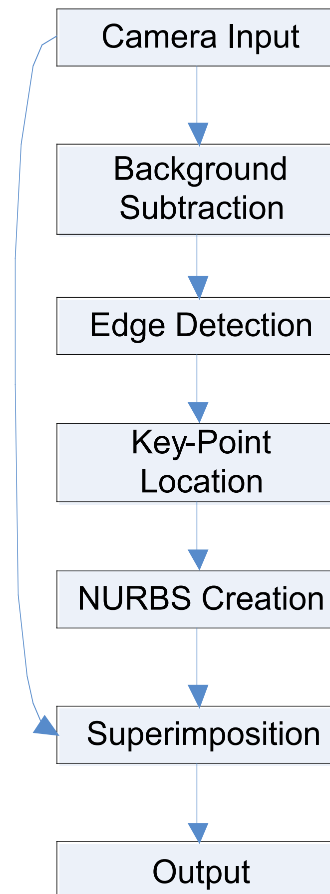


Figure 4.0.1 Flow chart of the process

4.1 Background Subtraction
(by Kosta Kaplanis)

The background subtraction step first consists of taking a snapshot of the background via a live feed. While the snapshot is being taken, the background video is converted to monochrome, having each frame of the background feed reproduced in tones of grey. This is done using the jit.rgb2luma[1] object, where one plane of luminance data is produced from a four plane input matrix (the background frame). According to the jit.rgb2luma help in Jitter, the luminance value is calculated by the formula

luma = alpha scale * a + red scale * r + greyscale * g + blue scale * b

where a, r, g and b are the different pixel values of the alpha, red, green and blue planes respectively. Each pixel value of each plane is multiplied by a specified scaling factor to vary the luminance of each plane. When no scaling factor is specified, Jitter assumes default scale values of alpha scale = 0, red scale = 0.299, green scale = 0.587 and blue scale = 0.114.

The resulting one plane monochrome matrix is then used in the process of calculating its mean value over time. Each pixel of the monochrome matrix has a certain value over time. The mean value for each pixel is taken by adding each value of the pixel over a certain time duration, and then dividing the result by the number of sample values. In other words, for a pixel p, if 5 different sample values where taken over time, then its mean value P_m over time would be

$$P_m = (P_1 + P_2 + P_3 + P_4 + P_5) / 5$$

The mean value of each pixel will be key in eliminating the background. Once the mean value matrix has been calculated and the output of this matrix has been captured,

[1] http://www.cycling74.com/documentation/jit.rgb2luma

the subject presents her/himself in front of the camera. This live feed is also converted to a greyscale monochrome output using the same method as the background snapshot procedure (using jit.rgb2luma).

The background elimination is accomplished by subtracting the mean value of each pixel of the mean value matrix (the background snapshot) from each pixel of the live feed matrix involving the person in front of the background.

The final step is using jit.sobel to determine the edges in the video. This object, as mentioned in the Cycling '74 documentation section, takes a matrix as input, and computes its spatial gradient using convolution kernels. It then brightens pixels in the image which have a "high spatial frequency" and darkens pixels which have low spatial frequencies. In other words, it brightens pixels that have neighbouring pixels of different values and darkens pixels that have neighbouring pixels in the same range of values. We chose the Sobel method because of its ability to detect edges in any direction, whereas the Prewitt method works mainly on horizontal and vertical edges. Our objective here was to manipulate the video in order for everything to be black, except the person's edges, which would be white. But just applying jit.sobel did not suffice, since we still had non-black pixels in the video that were not part of the person's body. These noise pixels needed to be eliminated. After examining the values of these noise pixels in the video matrix, we determined that they were all in the range [0;0.2]. We then used jit.op >= 0.3, which takes in a matrix, and only outputs any values superior to 0.3, setting these to 1 and the other values to 0. This eliminated the noise completely, keeping the edges of the body intact. The end result was a black background and white edges, describing the contour of the body. This result is illustrated in Figure 4.1.1. The patch is available in Appendix III.

Figure 4.1.1 Matrix after the background is eliminated and the edges are detected

4.2 Edge Detection: Aggressive

(by Mohannad El-Jayousi)

With the background eliminated and the subject isolated, determining where the subject is is the next hurdle. Team Trapped thought of a plan where the edge detection would be thorough and exact. This thoroughness would provide a precision that would allow for a web that can be manipulated in very subtle ways, and would follow the subject's contour intently.

The "aggressive" edge detection patch works as follows. The matrix outputted from the background subtraction patch is a 1-plane, greyscale 320x240 pixel matrix with a black background and a subject with a white contour. This contrast between a 0 RGB background and anything between 1 and 255 RGB subject is the key to detecting edges.

Since the aggressive patch is looking to achieve high precision, it was decided it would iterate pixel-by-pixel, row-by-row looking for edges. Quickly, Team Trapped determined it was unfeasible to attempt iterating through 76,800 pixels checking each

one's RGB, and still hoping to have a decent frame rate for the output. To overcome this barrier, the matrix is down sampled to a 32x24 pixel matrix, which would mean a much less cumbersome iteration through 768 pixels.

This iteration deals with looking for edges, but which of the subject's edges are going to be relevant to the web? Team Trapped concluded that having precise left and right edges would do the job. To determine left and right edges, the aggressive patch would therefore need to iterate from two different directions to find two sets of edges. The first iteration would start at the top-left corner, meaning the (0, 0) coordinate of the matrix, and work its way along the first row, incrementing the x coordinate by 1 and maintaining the y at 0. For each pixel it lands on, it compares the RGB value of the pixel to 0, and if the value returned is true, it determines the subject is not there and moves on to the next pixel. If however the RGB value is greater than 0 at a pixel, its (x, y) coordinate is stored in a data structure that has 24 storage slots, one for each row, to store the right edges. Similarly, there is another such matrix to store the set of left edge coordinates. Since there is only one storage slot for each row per side, when another non-zero RGB is found on the same row, the previously stored one is overwritten. This is not an issue, since for the iteration going from left-to-right for each row, it will always be the latest non-zero coordinate stored in the data structure, meaning it has the rightmost boundary coordinate for that row when it is done iterating through it. Once done with a row, the y is incremented by 1, and the x is reset to 0, ready to analyze the following row from left-to-right to find the rightmost edge. As this is happening, the second iteration does the same thing but starts from top-right, iterating right-to-left to store the leftmost edge coordinate of each row. See Figure 4.2.1 for an example of such a matrix.

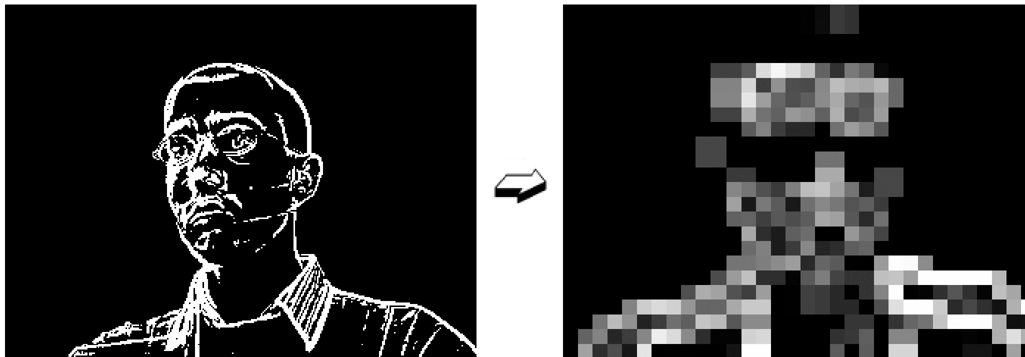


Figure 4.2.1 Down sampled matrix for edge detection iteration

One thing worth noting is that the rightmost coordinates have to be stored from last to first, contrary to the leftmost coordinates that are stored from first to last. This is because later in the web creation patch, the coordinates that are fed into the control points of the NURBS object must be passed in a counter-clockwise order. More precisely, counter-clockwise meaning from top-to-bottom and then bottom-to-top again, hence the backward order of the rightmost edge coordinates.

The entire iteration process comes to an end, resulting in two matrices storing all the right and left edge coordinates. If there are rows with nothing but pixels of RGB value 0, then a –1 is stored in the edge coordinate matrices to indicate it. The –1s are dealt with next.

The two 2x24 matrices storing the edges are glued into one using jit.glue[2], the left one with its top-to-bottom edges coming first, and the right one with its bottom-to-top edges coming second, forming a matrix that is 2x48. This matrix is then split into two matrices using jit.scissors[3], 1x48 each, to separate the x from the y coordinates. These matrices are then iterated through simultaneously, and whenever the x coordinate is not a

[2] http://www.cycling74.com/documentation/jit.glue
[3] http://www.cycling74.com/documentation/jit.scissors

-1, it is copied over to a "final destination" matrix (with the y coordinate in an adjacent matrix), and a counter is incremented to keep track of how many rows have an edge. This counter will be passed over for the web manipulation patch, as it will dynamically determine how many control points will be used (with a maximum of 48). Along with the counter, the x-coordinate and y-coordinate matrices are also outputted for the web manipulation patch's benefit.

With the process complete, it would then be time to have a clean slate for the next cycle of the process. The various matrices used along the way are reset, as are all the counters used to iterate through them. The entire cycle occurs at a rate set by a metronome, which was once every 500 ms.

Despite having the processing occur only twice every second, the patch was iterating through 768 pixels from two different directions (meaning 1536 different evaluations), making a comparison for each one, writing the coordinates of the pixel to a matrix every time it was found to have an RGB greater than 0, then the matrices are glued together and split into two columns, and evaluated simultaneously to remove "blank" rows. The processing overhead required to achieve all of this was relatively high, and it affected the performance of our patch. In an attempt to reduce overhead, the image was down sampled to a 16x12 matrix. Though this did slightly improve the frame rate, the patch was still slow, and the 16x12 down sample created problems of its own: if the subjected is not close enough to the camera, he or she may not appear accurately in the down sampled matrix. Secondly, the excessive down sampling causes for discontinuities in the subject's contour, which would throw off the control points of the NURBS web.

Unfortunately, due to the issues stated above, the aggressive edge detection patch had to be replaced for the final product. Had it made it in the final cut, the web would have reacted with a lot of precision to the subject's movement, and wrapped itself around the subject a lot more tightly. Ambitious as it was, the aggressive edge detection patch could not compete with the efficiency of the passive edge detection patch, which is documented in the next section. The patch is available for reference in Appendix IV.

4.3 Finding Key Point Coordinates
(by Olivier Leprince)

Once the background has been eliminated and the edges have been detected, an important part of the process is determining the location of key points on the subject's body. These key points will be used to determine the position of the web in the final output.

Our initial approach involved finding all possible edges on the person and using their coordinates to control the web. This was the aggressive edge detection method. By edge detection we mean that we are finding the location of each edge of the person and using those coordinates to control the web (not to be mistaken with the edge detection process from jit.sobel[4] which outputs the edges in the video). Although this method would have given an extremely accurate location of the edges, certain drawbacks such as processing time made it difficult to implement.

The method we used involved finding the location of fewer key points in the live input. This made the computation lighter and therefore faster to process. This method

[4] http://www.cycling74.com/documentation/jit.sobel

uses the object jit.findbounds[5] that comes with Jitter. This object, when fed a matrix containing video information, allows us to determine the boundaries of certain colors in the video. The object iterates through the matrix and outputs the coordinates of the minimum and maximum points in the matrix containing the color value we are looking for. Since at this step of the process, the video is composed of black pixels for the background and white pixels for the edges of the person, we used jit.findbounds to determine the bounding box for white pixels. Using the coordinates of the first and last white pixels, we were then able to determine the rectangular portion of the video that contained the person.

We use this rectangular portion of the original matrix and place it in a new matrix, using jit.submatrix[6]. This object allows us to create a new matrix that represents that bounding box, using the first white pixel's coordinates to determine the starting point of the sub matrix (offset) and the difference between the last white pixel's coordinates and the first to determine the dimensions of this new matrix. See Figure 4.3.1 for an example of such a matrix:

Figure 4.3.1 Bounding box submatrix

[5] http://www.cycling74.com/documentation/jit.findbounds
[6] http://www.cycling74.com/documentation/jit.submatrix

Once this matrix containing the boundaries of the person has been created, we divide it into four matrices using jit.scissors. This object allows us to divide any matrix into the number of rows and columns we want. We divided the bounding box into 2 rows and 2 columns, yielding four equally sized matrices. These four matrices represent each quadrant of the bounding box. The first matrix represents the top left part of the bounding box, the second represents the top right part, the third and fourth represent the bottom left and bottom right parts respectively.

We apply jit.findbounds again, on each one of these quadrants to determine the more accurate location of the boundaries, which will be used to control the web. The basic principle is the following:

To understand the process, a brief overview of the web creation is required. The web has been created using NURBS, which will be described in the next section. Basically, the web is created using points, called control points that determine what the shape of the web is going to look like. We have used five control points, the first one controlling the upper middle part of the web, the second controlling the top left of the web, the third, the bottom left, the fourth, the bottom right and the fifth controlling the upper right part of the web.

Using the location of the boundaries in each quadrant of the bounding box, we use the upper left quadrant to determine the upper left control point's coordinates, the upper right quadrant to determine the upper right control point's coordinates, the lower left quadrant for the lower left control point's coordinates and the lower right quadrant for the lower right control point's coordinates.

Of course, in each quadrant, not all the information received from jit.findbounds is relevant. For example, in the upper left quadrant of the bounding box, the point of interest, that will be used to control the top left control point, is the leftmost and up most boundary, corresponding to the first white pixel's coordinates in that particular quadrant. For the top right quadrant, it is the first white pixel's horizontal coordinate and the last white pixel's vertical coordinate that determine the top right control point's coordinates.

For the bottom left quadrant, it is the first white pixel's horizontal coordinate and the last white pixel's vertical coordinate that determine the bottom left control point's coordinates.

Finally, for the bottom right quadrant, it is the last white pixel's coordinates, horizontal and vertical, that control the bottom right control point.

The final step in this process is to translate and scale the pixel coordinates to correspond with the web control point's coordinates.

This is due to the fact that the video matrix has coordinates relative to its origin, which is the top left cell of the matrix. But when drawing in OpenGL, the origin of the axis is located in the center of the screen. That is why a translation is needed.

To translate the coordinates, the first step is to determine the coordinates of the key points found in each quadrant with regard to the original matrix, which contains the whole video. Since the coordinates of the key points are given with regard to each quadrant's dimensions, the horizontal and vertical offsets of the bounding box have to be taken into account.

For the first quadrant, which is the top left part of the bounding box, the coordinates of the key point will be its coordinates in that quadrant plus the offsets of the bounding box with regard to the original matrix.

For the second quadrant, which is the upper right part of the bounding box, the same offset has to be added for the horizontal and vertical coordinates, but also the horizontal offset of that quadrant's origin with regard to the bounding box's origin. Since the four quadrants have the same dimensions (bounding box divided into four equal regions), this offset corresponds to half of the bounding box's horizontal dimension.

For the third quadrant, which is the bottom left part of the bounding box, the same offsets have to be used with regard to the original video matrix. Since this quadrant is located in the bottom half of the bounding box, the vertical offset with regard to that bounding box has to be taken into account. This corresponds to half of the vertical dimension of the bounding box.

And for the fourth quadrant, located in the bottom right part, the offset from the original matrix needs to be added and the offset with regard to the bounding box also. In this case, that offset corresponds to half of the dimensions of the box applied on both the horizontal and vertical coordinates. See Figure 4.3.2.

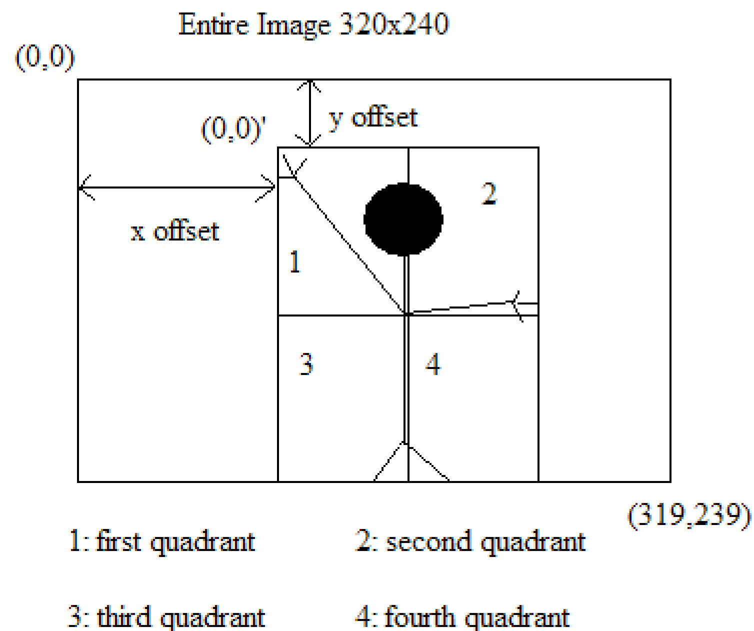


Figure 4.3.2 Depiction of the bounding box and quadrants

This gives us the location of the key points with regard to the origin of the original video matrix.

Once the position of the key points have been determined with regard to the original matrix, we need to convert these with regard to the origin of the OpenGL drawing environment, located in the center of the screen.

In order to do this, we use the dimensions of the original matrix, which is a 320 by 240 matrix, to place the origin in the center of that matrix. For the horizontal coordinates, subtracting the key point coordinate by half the horizontal dimension of the original matrix yields its horizontal coordinate. For example, if a key point's x-coordinate is 1 with regard to the original matrix, which would mean that it is located on the top left part

of the image, subtracting by 320/2 = 160 would result in the x coordinate being equal to -160 with regard to the center of the matrix. If it was 160, meaning it was located at the center of the original matrix, subtracting 160 would make it equal to 0, which is indeed the center of the matrix.

A similar operation is applied to the vertical coordinates, the only differences being that we subtract half of the vertical dimension, which is 240/2 = 120 and that, since anything in the upper half of the matrix is positive and anything in the lower half needs to be negative, we have to multiply the coordinate by -1. Once we have multiplied the vertical coordinates by -1, all points in the upper half of the matrix will be positive and all points in the lower half will be negative.

The last step of this process involves scaling the coordinates to our rendering environment. We have decided that the bounds for the location of the control points of the web will all be in the interval [-1;1]. Anything outside this boundary will be off screen and won't be visible.

Up to now, the coordinates of the control points range from -160 to 160 on the horizontal axis and from -120 to 120 on the vertical axis. In order to have only coordinates between -1 and 1, we simply divide the horizontal coordinates by 160 and the vertical coordinates by 120.

Now the coordinates of the key points all range between -1 and 1 and can be sent to the next patch which uses these coordinates to place the control points of the NURBS surface which represents the web.

The principle of using the bounding box and dividing it into four sub-regions resembles a data structure used for dividing two dimensional spaces called a Quad Tree, even though it is not as complex and is not recursive.

These data structures are based on the "divide and conquer" method, where the tree is defined by its root which contains the entire image, and has four children, which are sub-regions of the image, which themselves have four children, and so on. Each sub-region contains specific information on that particular part of the image. These data structures are especially helpful for image manipulation such as compression and locating specific pixels in an image. We were not aware that such a structure could be applied to image manipulation, and only found out after explaining the method to a colleague that has a background in computer graphics and data structures. We then made research on the subject to compare the two.[7]

Our approach applies a similar principle, where the root would be the bounding box and the quadrants would be the four children. Taking this method and applying it even further would have made the location of key points more accurate, but would have required dividing each quadrant into four new quadrants, using only significant ones (the outermost quadrants) to retrieve pixel coordinates. This would also have required additional control points as well.

The main advantage of this method is that it requires less computation, making it easier to implement in real-time situations.

The only issue with this approach is that in certain situations, the jit.findbounds will output -1s as coordinates. This occurs when no white pixel has been found in the

[7] Carbonetto, Peter. 1999. Picture Representation Using Quad Trees. http://www.cs.ubc.ca/~pcarbo/cs251/welcome.html

region we are analysing. This would happen if a quadrant of the bounding box did not contain any white pixels.

Of course, computing the translated and scaled coordinates using these -1 values makes no sense and yields false coordinates. After testing the resulting aspect of the web in these specific situations, we determined that it was not an issue since the overall effect of the web following the person was intact. This patch can be seen in Appendix V.

4.4 NURBS - Web Creation
(by Olivier Leprince)

As briefly mentioned above, the web was created using NURBS (Non-Uniform Rational B-Splines). These can be created using the jit.gl.nurbs[8] object that comes with Jitter. The basic principle is that a surface can be created using points. These are used to generate complex surfaces simply by placing control points that describe the overall shape of the surface. These control points are actually used to create a curve that will "follow" these points. In order to obtain smooth and continuous curves, interpolation on these control points is needed. The order of interpolation can be changed to obtain different curvature with regard to the control points. The fact that the curve is smooth and continuous means that it won't necessarily "pass" through the control points but will be "attracted" to these.

An important aspect of generating these surfaces is the order in which the points are placed. Since the curves will start at control point 1, pass by control point 2 and so on, it is important to keep the order of the points to obtain a continuous curve. Furthermore,

[8] http://www.cycling74.com/documentation/jit.gl.nurbs

in order to obtain a closed surface, we need to make sure the first and last control points have the same coordinates.

Another important parameter in controlling NURBS surfaces is the weight of the control points. By changing the weight of each point, we can alter the impact each specific control point has on the overall curve. For example, a weight of 0.5 for a specific control point will make that control point “attract” the curve and a weight of 1.5 will “repulse” the curve away from the point.[9]

In our project, we wanted a closed surface, which would look like a web. In order to do so, we placed the control points in a specific order. The first one is located at the top middle part of the screen, the second on the left top part, the third on the bottom left, the fourth on the bottom right and the fifth on the top right of the screen. We also placed other control points, one of which has the same coordinates as the first point, in order to obtain a closed surface. The rest of the control points were placed at the origin. By drawing the surface in wire frame, we obtained curves describing the overall shape of the surface and lines originating from the center control points. We used the wire frame to give the surface a web-like appearance. By changing the dimension of the grid, we can vary the number of “wires” appearing inside the surface, therefore changing the web’s thickness. See Figure 4.4.1.

[9] Lavoie, Philippe. 1999. An Introduction to NURBS. http://libnurbs.sourceforge.net/nurbsintro.pdf

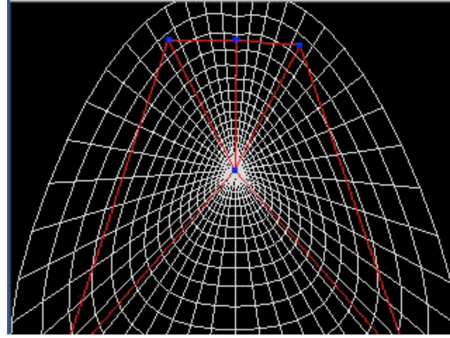


Figure 4.4.1 NURBS object with control points visible

Once the coordinates of the key points of the person have been determined, they are sent to the NurbsControl patcher. This patcher takes the coordinates of the top left quadrant key point and associates it with the top left control point. The bottom left quadrant's key point is associated to the bottom left control point, the bottom right with the bottom right control point and the top right quadrant with the top right control point. The first control point's coordinates are determined by the bounding box's minimal coordinate for the vertical axis and the middle point of the bounding box for the horizontal coordinate. This makes the control points follow the key points in the video.

But, as mentioned above, the curves will not necessarily pass by all the control points. That is why we change the weight of the control points to make them "attract" the curve, making the curve overlap the control points, thus overlapping the person in the video. We set all the left and right control points as having weights of 0.6, to make sure the surface wraps around the person. The middle point has a weight of 0.8, reducing its impact on the overall shape of the web.

The jit.gl.nurbs object receives a 4-plane matrix, containing the coordinates of all the control points. The first plane corresponds to the x coordinates of each control point, the second contains the y coordinates, the third contains the z coordinates, and the last

plane contains the weight of the points. When receiving the coordinates, we place those into a matrix and send that matrix to the jit.gl.nurbs object, which renders the web. We did not use the z coordinates of the control points because our main objective was to make sure that the web manipulation in 2-D was working. If we had more time, we could have either made the web have ripples, or create a depth effect with regard to the person, which would have required depth analysis on the live feed.

Once the web was created, we needed to combine the live input and the web to produce our final output. For some reason, we were not able to place the rendered web into a Jitter matrix that we could manipulate like any other Jitter matrix. After several unsuccessful attempts, we avoided the problem by mapping the live input onto a Video plane located behind the web using the jit.gl.videoplane[10] object. See Figure 4.4.2. To see the patch in its entirety, refer to Appendix VI.

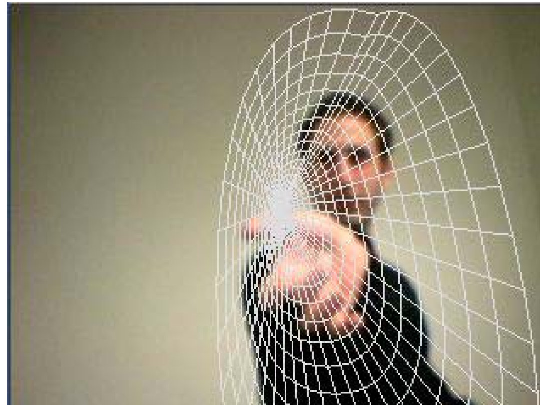


Figure 4.4.2 NURBS object superimposed over live feed

[10] http://www.cycling74.com/documentation/jit.gl.videoplane

4.5 Trapped Patch

The Trapped Patch is the all-encompassing driver of the other patches detailed previously in this section. In other words, it is a central manager. See Appendix II.

4.6 Constraints and Challenges

During the development of Trapped, the team had to face and overcome various challenges. Some of these include several failed approaches to background subtraction, the cumbersome aggressive edge detection patch documented in section 4.2. Since this patch was scrapped, it affected adjacent patches, mainly the NurbsControl patch since the latter expected 48 control points from the former, but had to be adjusted to accept only 5. Other challenges revolved around the NURBS object not easily mixable with a Jitter matrix, forcing the team to look for alternative methods to perform this feat. Due to the fact that the final output was not purely a matrix, recording the output was a challenge in itself, and ultimately caused for recorded output to be slightly jittery (no pun intended) as can be seen in the final video montage available for viewing at hybrid.concordia.ca/~mohan_el (attention: this choppiness issue only occurs to *recorded* output, the output itself is in no way choppy, as witnessed during live presentation). Finally, the time constraint did not allow for adding texture to the web, though it would have been fairly simple to achieve this.

5.0 MILESTONES AND DELIVERABLES

5.1 Project Lifespan/ Milestones

The project lifespan was approximately eight weeks. Throughout that time, the team's effort went into various activities, often times the activities were performed simultaneously. The project stages are illustrated below in Figure 5.1.1 as a Gantt chart.

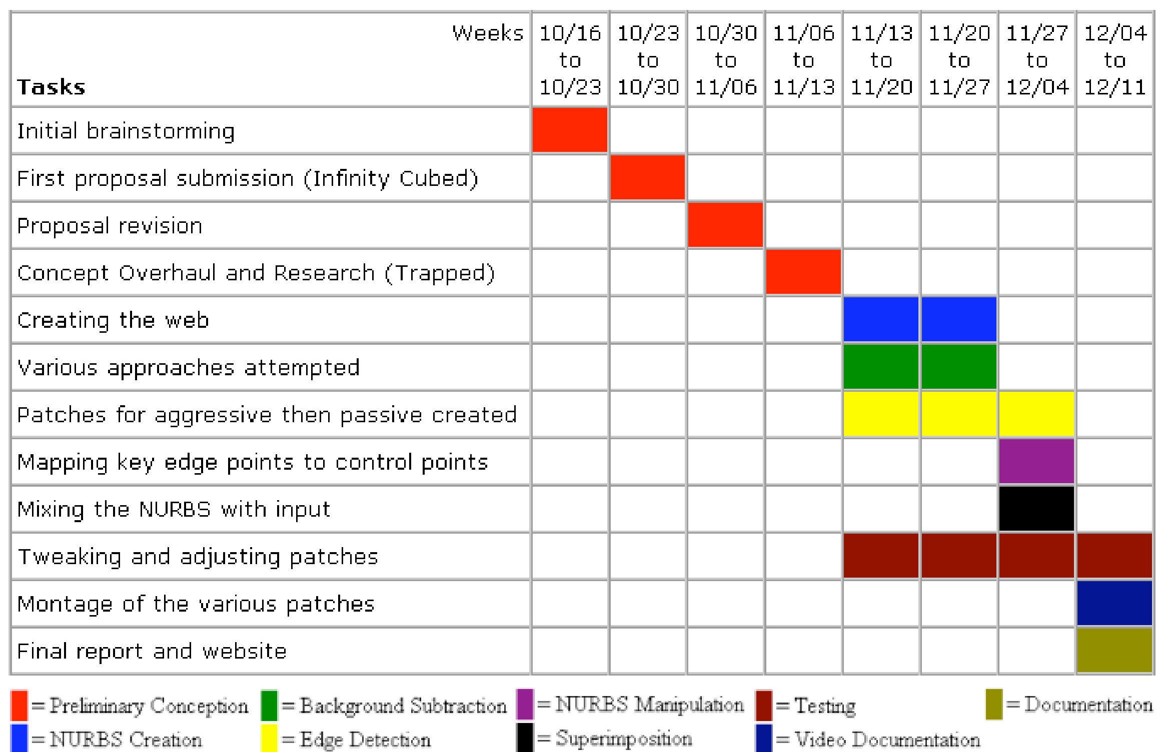


Figure 5.1.1 Gantt chart illustrating project lifespan

5.2 Deliverables

Milestones came at the end of the different activities on the Gantt chart illustrated in Figure 5.1.1, and along with these milestones, a deliverable was usually created. Refer below to a list of the deliverables.

- ***Initial concept proposal:*** October 23, 2006
- ***NurbsControl patch:*** November 26, 2006
- ***Aggressive Edge Detection patch:*** November 26, 2006
- ***BackGround patch:*** November 26, 2006
- ***Technical Powerpoint presentation:*** November 27, 2006
- ***FindCoords patch:*** December 4, 2006
- ***Trapped patch:*** December 7, 2006
- ***Video montage:*** December 8, 2006
- ***Official website:*** December 11, 2006
- ***Final report:*** December 11, 2006

6.0 CONCLUSION

Trapped has been a very challenging project, and all the members of the team feel a sense of accomplishment due to the high learning curve of the tools that were used, and the various obstacles that had to be overcome over the course of the project's lifespan. The final result makes all of us proud, and we hope it will generate interest in audiences that see it in action, whether they be familiar with real-time video processing or not.

REFERENCES

1. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.rgb2luma

2. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.glue

3. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.scissors

4. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.sobel

5. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.findbounds

6. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.submatrix

7. Carbonetto, Peter. 1999. Picture Representation Using Quad Trees. http://www.cs.ubc.ca/~pcarbo/cs251/welcome.html

8. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.gl.nurbs

9. Lavoie, Philippe. 1999. An Introduction to NURBS.

   http://libnurbs.sourceforge.net/nurbsintro.pdf

10. Cycling '74, 2006. http://www.cycling74.com/documentation/jit.gl.videoplane

LIST OF FIGURES

Fig 3.1.1 - Initial proposal illustration depicting envisioned output/installation.............5

Fig 3.2.1 - Spider web concept depiction...6

Fig 4.0.1 - Flow chart of the process...8

Fig 4.1.1 - Matrix after the background is eliminated...11

Fig 4.2.1 - Down sampled matrix for edge detection iteration.......................................13

Fig 4.3.1 - Bounding box submatrix 16

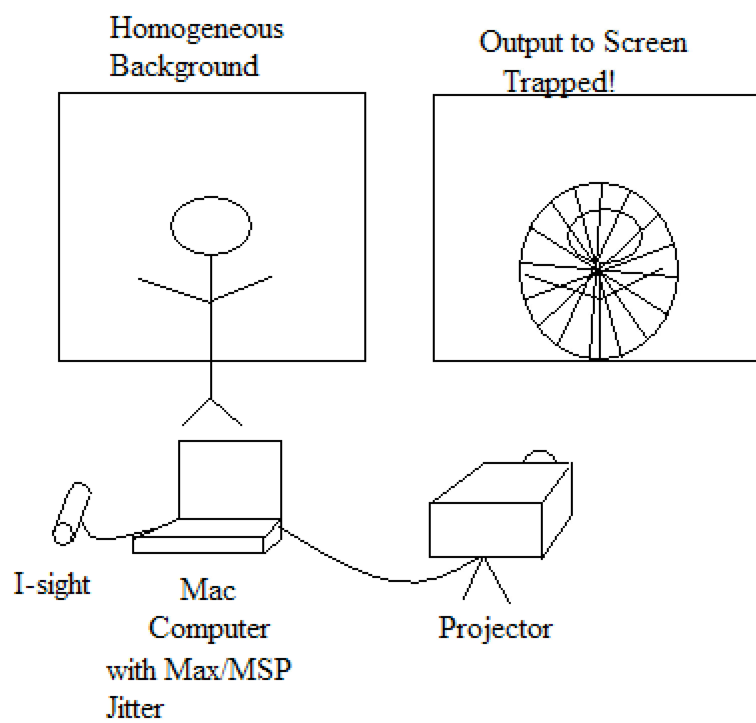
Fig 4.3.2 - Depiction of the bounding box and quadrants..20

Fig 4.4.1 - NURBS object with control points visible 25

Fig 4.4.2 - NURBS object superimposed over live feed..26

Fig 5.1.1 - Gantt chart illustrating project lifespan..28

APPENDIX I – INSTALLATION



APPENDIX II – TRAPPED PATCH

ON/OFF

open close Turn Camera ON/OFF

Reset Background

BackGround

FindCoords

Trapped

FULLSCREEN
PRESS "ESC"

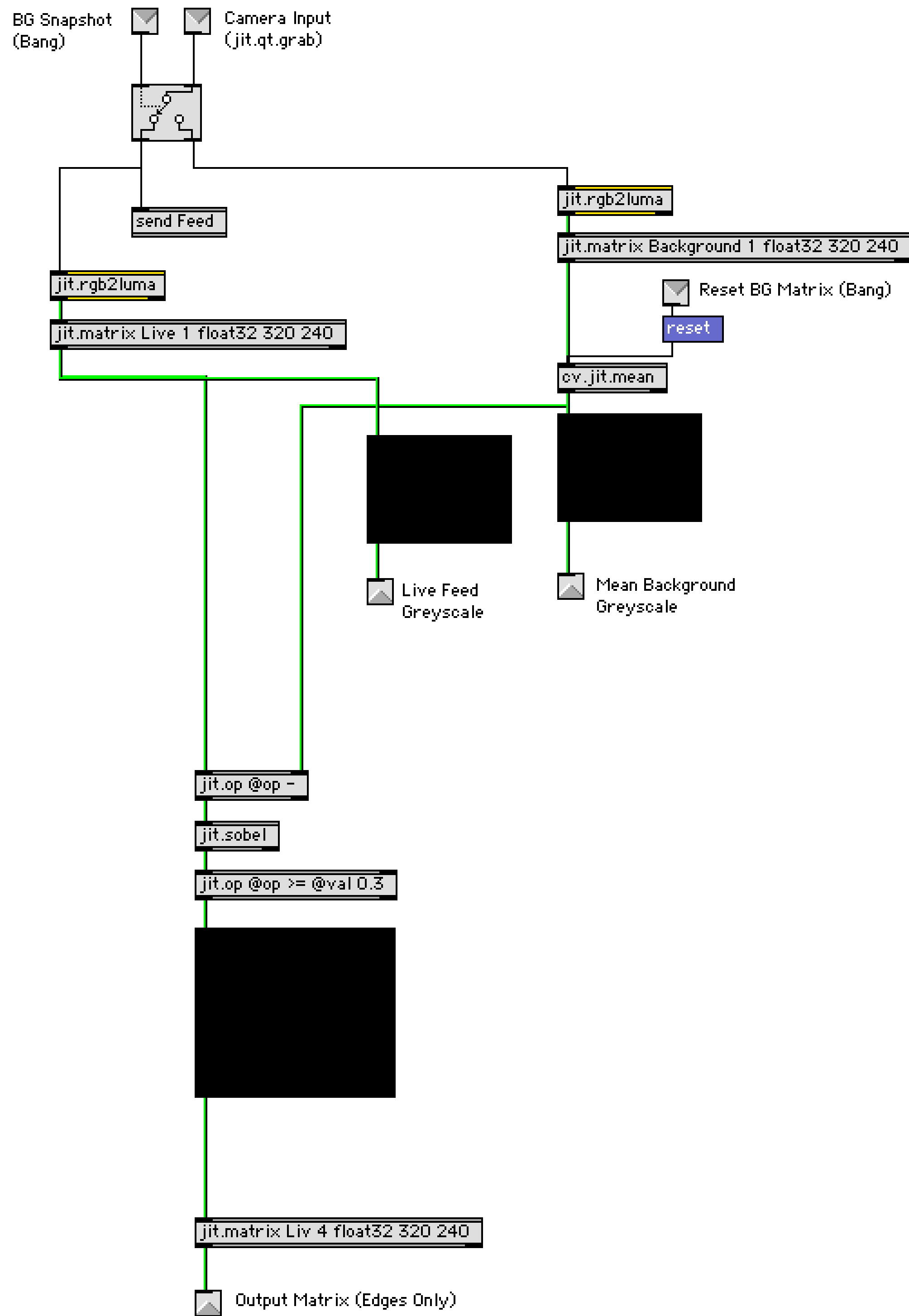
receive X1

0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

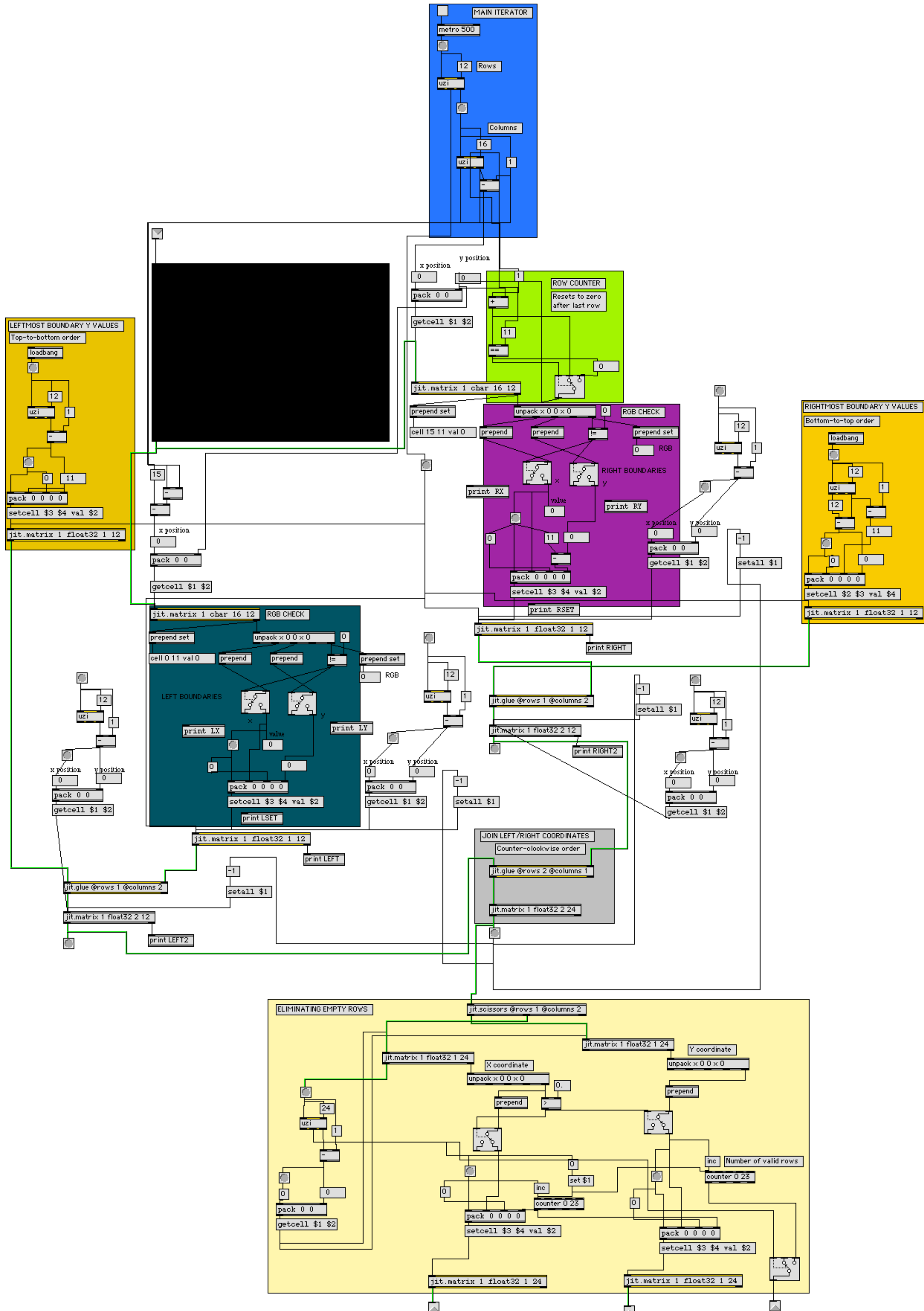
NurbsControl

WEB GENERATING

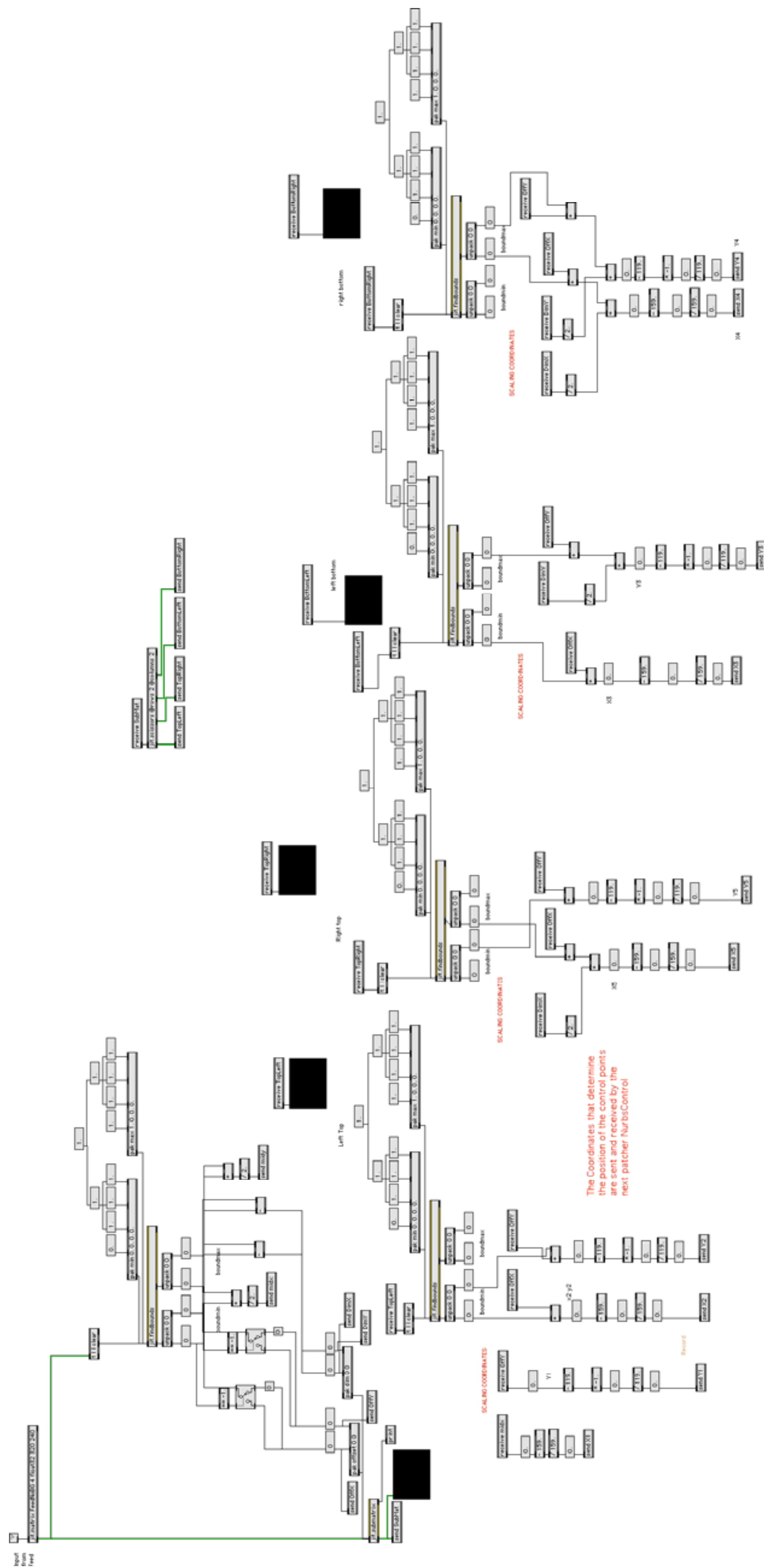
APPENDIX III – BACKGROUND PATCH



APPENDIX IV – EDGE DETECTION (AGGRESSIVE) PATCH



APPENDIX V – FINDCOORDS PATCH



APPENDIX VI – NURBSCONTROL PATCH