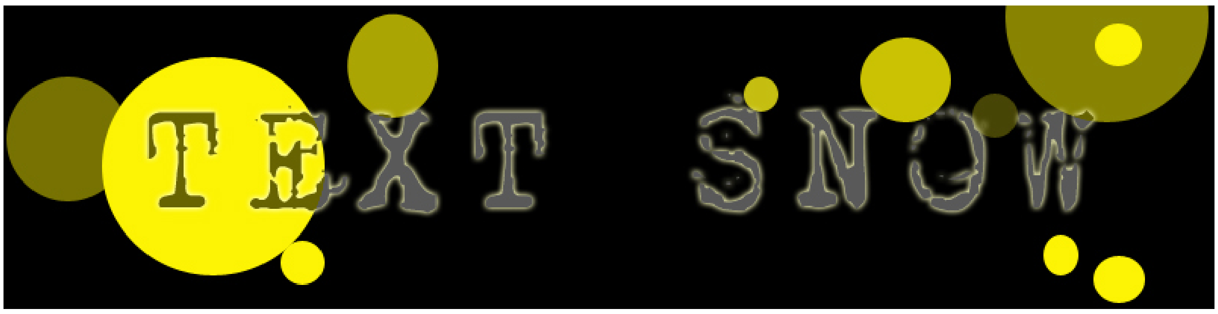


COMP 471 & CART 498C

Text Snow Project Report

Phuong Thanh NGUYEN, Elena ZAPEVALOVA, Philippe ADIB, Pierre NOGUES

December 11, 2006

Team Members

Phuong Thanh NGUYEN :	Jitter programming, Design, Set Construction
Elena ZAPEVALOVA :	Research
Philippe ADIB :	Jitter programming, Physics, Maths
Pierre NOGUES :	Jitter programming, Physics, Maths

Contents

1 Concept

Text Snowing is an interactive video installation art which requires the motion of the bodies to lift and play with falling letters that do not really exist. On a large projection screen, there are slowly falling letters from the top of the screen down to the bottom. These letters will stop falling when they meet any obstacle, for example: bodies, any object and keep falling when we remove these obstacles. When the letters land on any object, each character produces its specific sound. Sometimes, the participants can catch the whole word or phrase. These letters are not random because they form lines of a poem or sentence.

The idea of this project is exploring the interactive between real people (or object) and computer through a camera using motion tracking and video effect.

1.1 Description

"..Interaction between the viewers/performers and the text creates a unique dynamic suited for both individual and collaborative exploration. Viewers work together to decipher the poem, communicating with each other through their projected/mediated image, or alternatively, disrupt the "reading" of the poem by stealing letters from one another. One's image is inserted into a flat abstract space along with the text, while the text acts as objects that respond to forces in the real world and also to the physical gestures of viewers. Just as one's body is "dematerialized" onto the projected screen, the text is "materialized," appearing as substances that respond to physical movement. The text "continues to serve its symbolic function as an decipherable code, but also as an 'object' viewers can engage with as if it were a real physical entity [...] the physical act of catching letters is necessary in order to read the text at all [...] Because most of one's body is visible in the virtual space of the screen as well as in the physical space in front of the screen, a pleasurable confusion results between the screen space and the real space..."

2 Diagram and Flowchart

Here is the diagram of installation :

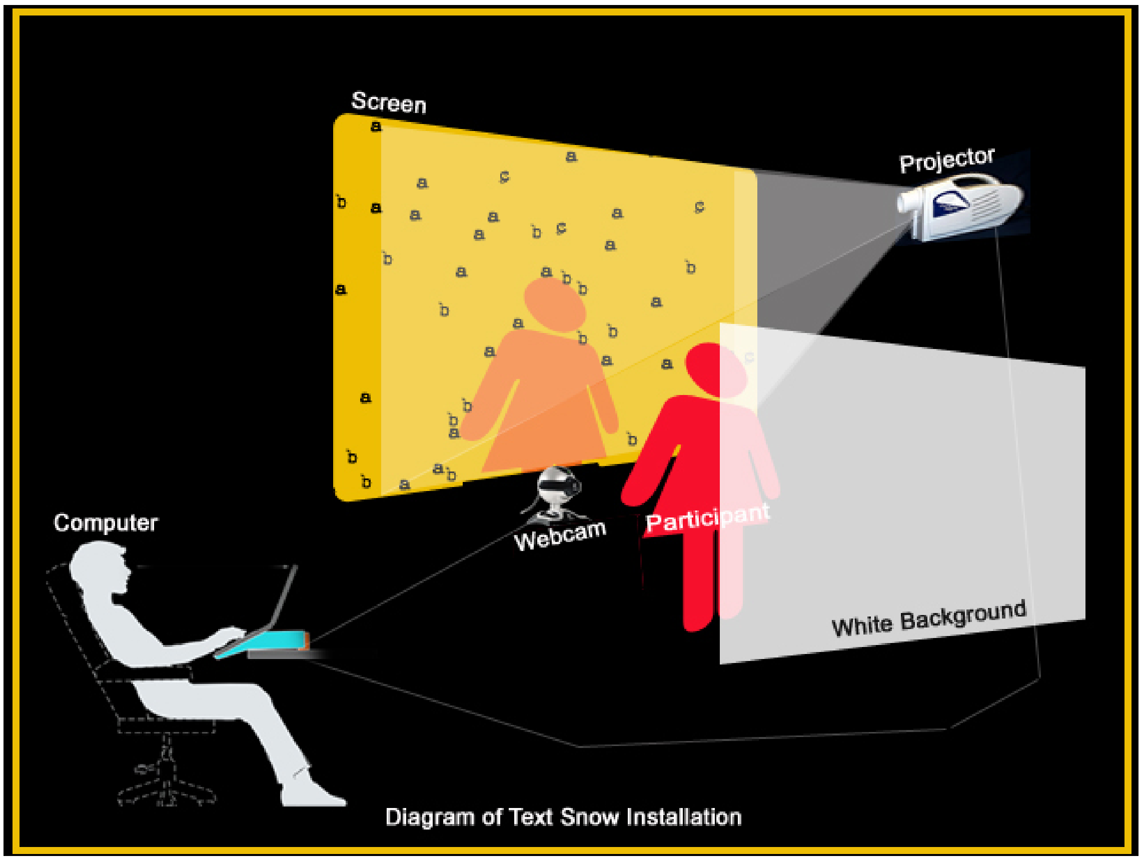


Figure 1: Installation diagram

And the Flowchart of the application :

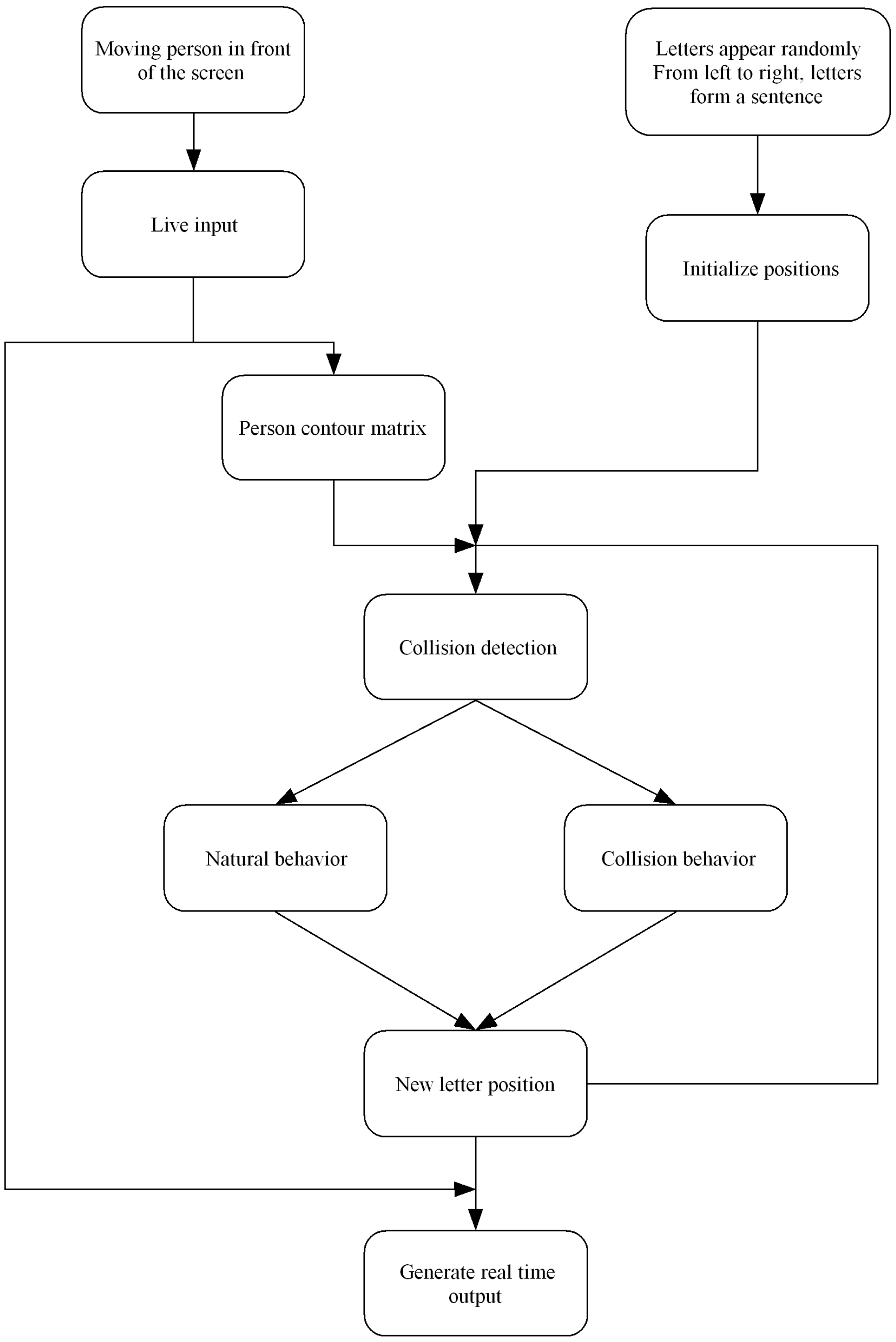
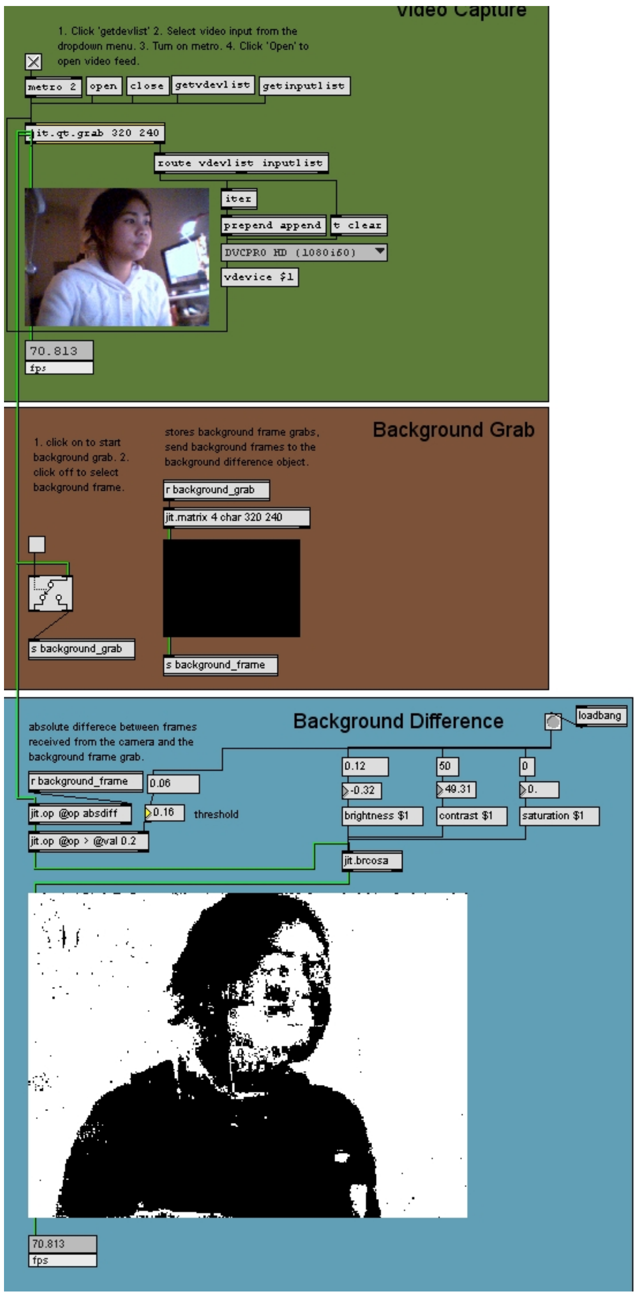

Moving person in front of the screen
Letters appear randomly From left to right, letters form a sentence
Live input
Initialize positions
Person contour matrix
Collision detection
Natural behavior
Collision behavior
New letter position
Generate real time output

Figure 2: Flowchart

3 Technical Description

3.1 Background subtraction patch

Author : Phuong Thanh NGUYEN



1. Click 'getdevlist' 2. Select video input from the dropdown menu. 3. Turn on metro. 4. Click 'Open' to open video feed.

1. click on to start background grab. 2. click off to select background frame.

stores background frame grabs, send background frames to the background difference object.

absolute difference between frames received from the camera and the background frame grab.

Figure 3: Background subtraction

General Algorithm

Background subtraction is a technique that enables a detection of the body shape of people or other objects in a scene in order to distinguish the pixels which belong to them from those that do not.

Mathematic Analysis

The arithmetic means:

- Background Subtraction frame by frame :
  Result matrix M = The Input matrix (static image) - the set of input matrices (frames over time)

- The Threshold :
  T: fix threshold value Result matrix after threshold = 0 if result matrix M at coordinate (x, y) $\leq$ T, and = 255 if result matrix M at coordinate (x, y) $>$ T

Apply algorithm into Jitter

Real time input through a webcam will be captured by jit.qt.grab. To subtract the background, a static image of the first frame is taken and input into a matrix. That frame image will be compared with the rest of frames by using jit.op @op absdiff to subtract the absolute different values between the first frame and the rest. The background value subtracted will be changed to black and the different value pixels will turn white or be inverted by controlling the threshold parameter of jit.op @op $>$. The black and white threshold output will go through jit.brcosa to adjust the brightness/contrast/saturation. Fina(lly, its output passes the object jit.convolve to reduce the noise of the shape and background.

3.2 Main patch

Author : Elena ZAPEVALOVA

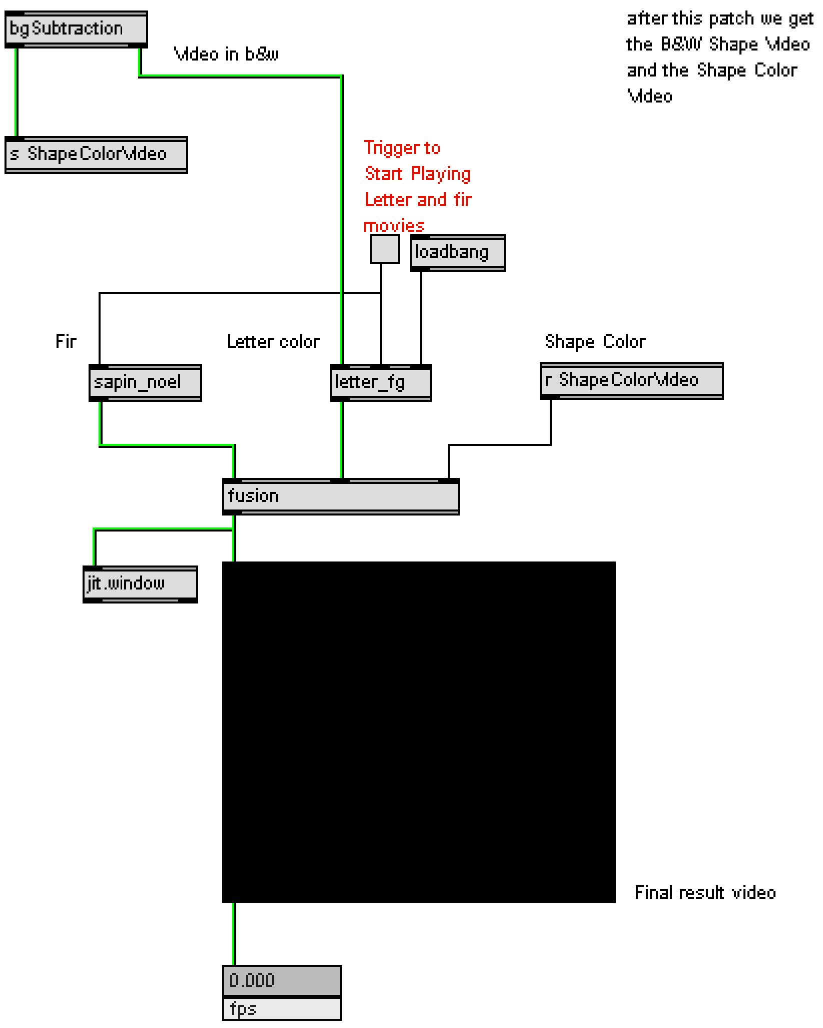


Figure 4: Main patch

In this patch we create a fusion of three images. The first one is the backgroud image with a Christmas theme. The second one is the image of falling letters and the third one is the live video image. To produce the image of falling letters we take the output from the bgSubtraction patch, which gives us the live video image transformed in black and white, and pass it through the patch letter_fg, which give us the letters falling with appropriate speed. All these three images pass as input to the patch fusion, which produces the final output image.

3.3 Sapin_noel & letter_movie_player patches

Author : Elena ZAPEVALOVA

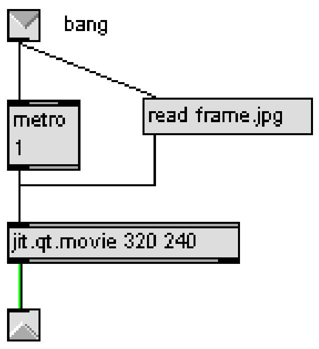


Figure 5: Sapin_noel patch

In this patch we use jit.qt.movie object which opens our background image and play it.

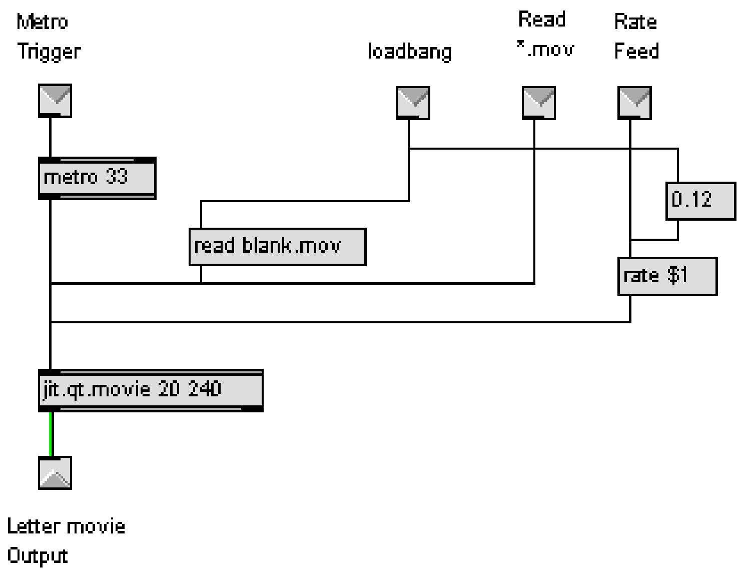


Figure 6: Letter_movie_player patch

This patch plays a letter, taking into account the speed calculated after collision process.

3.4 Fusion patch

Author : Phuong Thanh NGUYEN

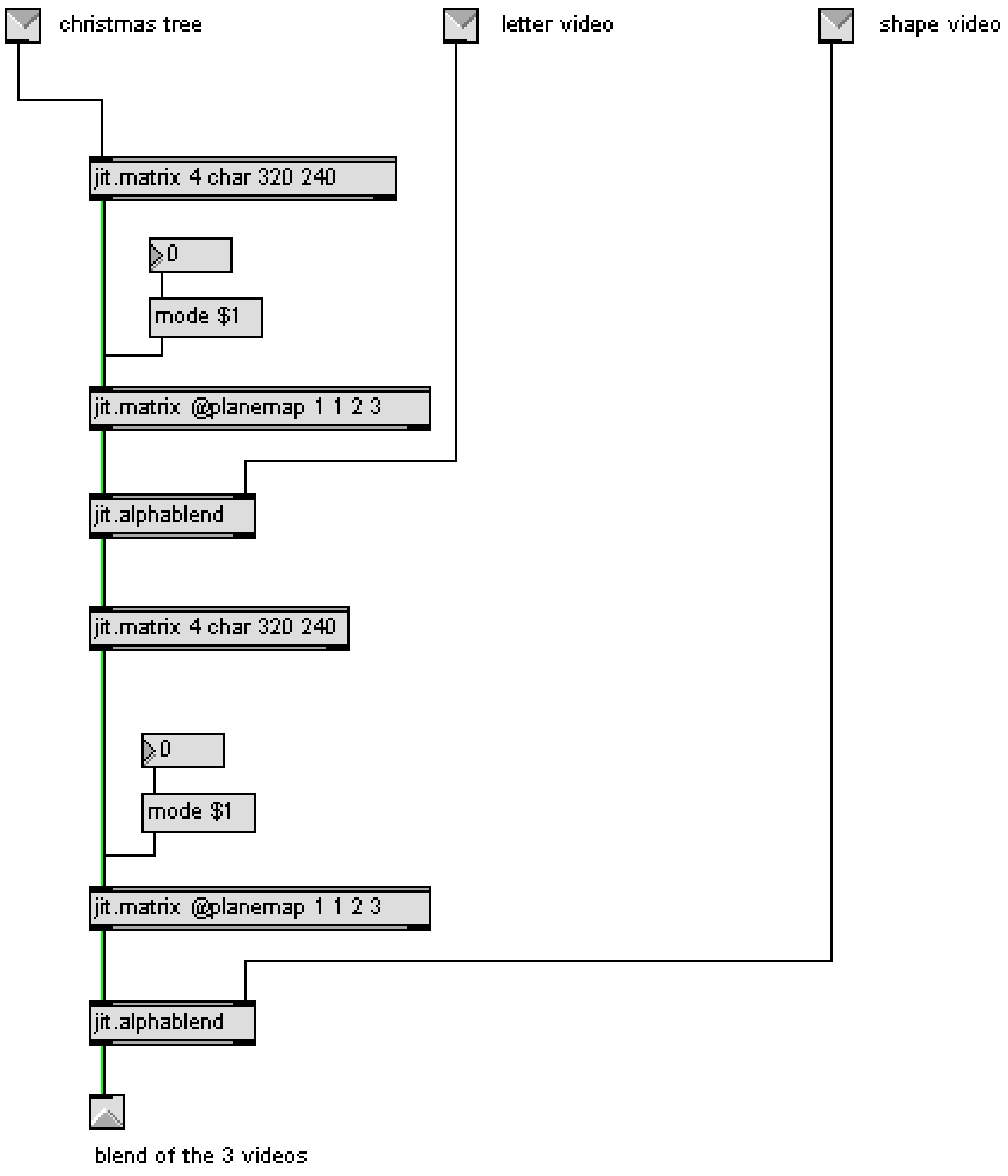

christmas tree
letter video
shape video
jit.matrix 4 char 320 240
0
mode $1
jit.matrix @planemap 1 1 2 3
jit.alphablend
jit.matrix 4 char 320 240
0
mode $1
jit.matrix @planemap 1 1 2 3
jit.alphablend
blend of the 3 videos

Figure 7: fusion patch

General Algorithm

Alpha Blending technique permits the combining of the alpha channels. Frames can overlay other layers in an image or video in order to show the transparent effects. The alpha channel is an additional eight bits used with each pixel in a 32-bit graphics system that can represent 256 levels of translucency.

Black and White: Opaque, fully transparent Gray: levels of translucency

Alpha Blending can be accomplished in computer graphics by blending each pixel from the first source image with the corresponding pixel in the second source image.

Mathematic Analysis

Equation of alpha blending:

Final pixel = alpha * (First image's source pixel) + (1.0-alpha) * (Second image's source pixel)

Apply algorithm into Jitter

The jit.alphablend object uses the alpha channel (plane 0) of the input matrix of one image or video in the left inlet as a per-cell cross fade value, and cross fades between the input matrices in the left and right inlets. In mode 0, a low value means more of the right input matrix, while a high value means more of the left input matrix. In mode 1, a low value means more of the left input matrix, while a high value means more of the right input matrix.

In the patch fusion we create a fusion of three images. As we cannot do three images at once, we first take two images and mix them using the jit.alphablend object. Afterwards we take this output and mix it with the third image in the same manner.

3.5 Letter_fg patch

Author : Elena ZAPEVALOVA

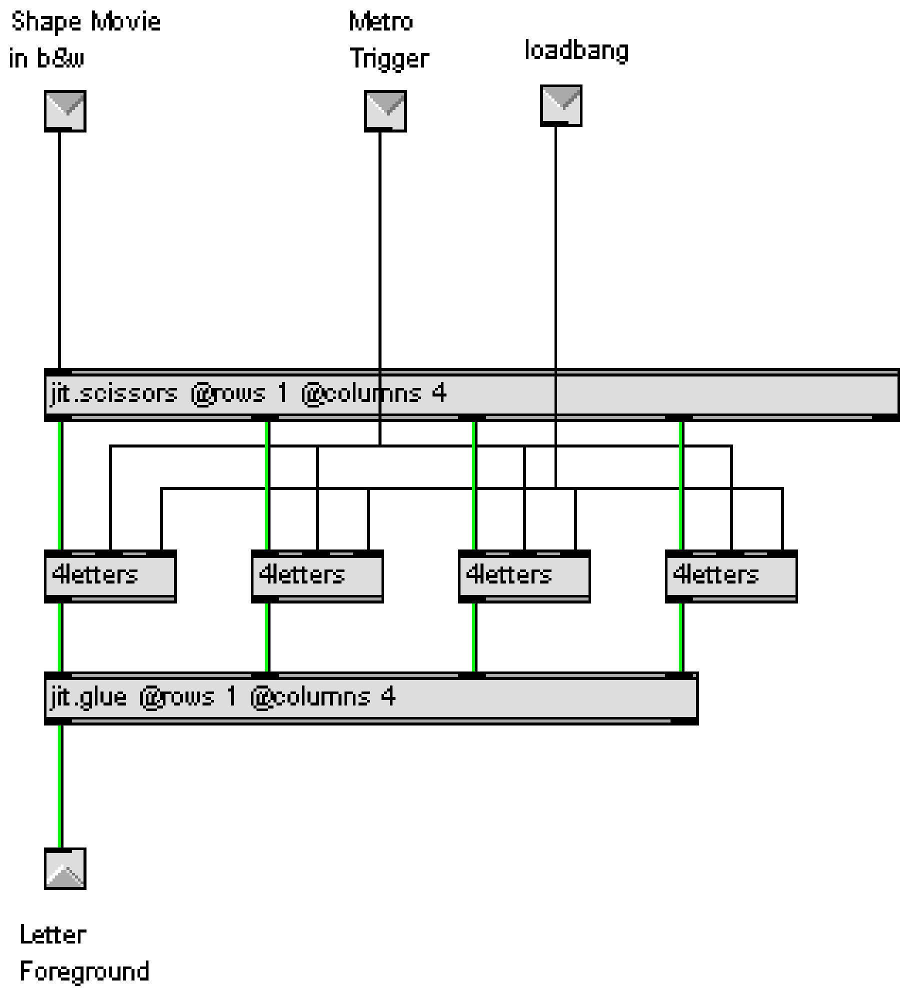


Figure 8: Letter_fg patch

This patch receives as input the live video transformed in black and white by bgSubtraction. We use the Jitter object scissors to cut this image into four columns. After this every column passes as an input to the patch 4letters, which cuts them again in four parts, and after that we paste the 4 outputs together (4 letters, see below) using Jitter object glue. We chose to cut the video in two times for readability reasons.

In order to end up with matrices of the size of a letter video, we used the maximum number of column divisions on a 320x240 matrix, that is, 16 new matrices, and no row subdivisions. This resulted in matrices of width $320/16 = 20$ and of height 240, that is, 20x240.

3.6 4letters patch

Author : Elena ZAPEVALOVA

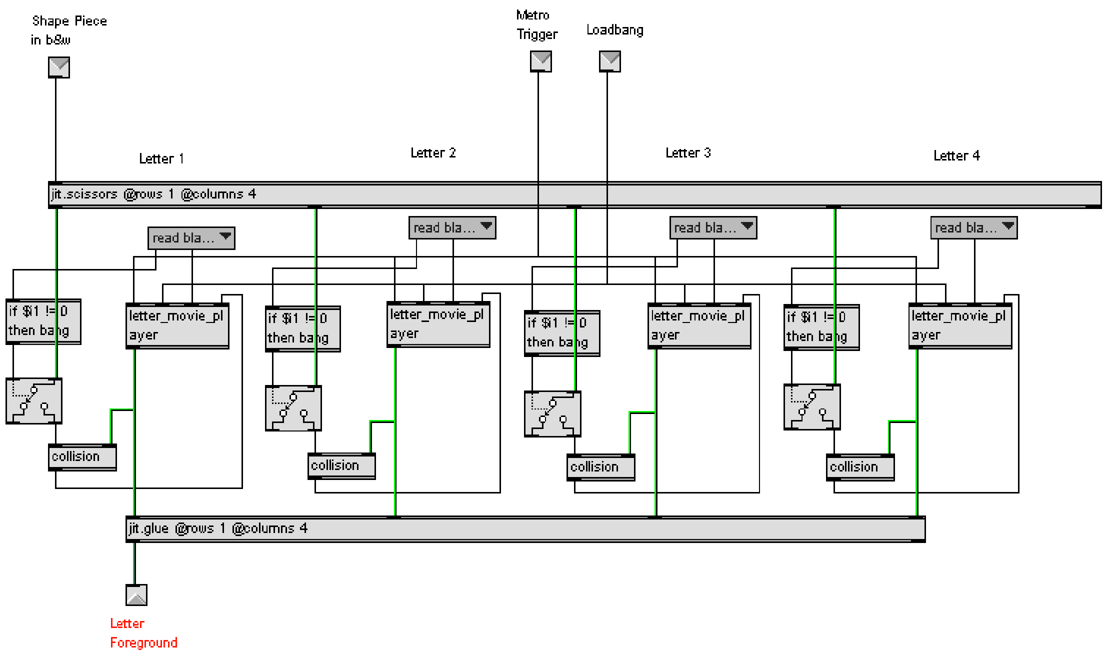


Figure 9: 4letters patch

This patch receives as input a part (one column) of the live video in black and white. We cut this image again in four columns using jit.scissors. Every column passes to the right outlet of the gate object. For every column we have also a video of falling letter. We pass this video through the patch collision to determine the collision and the rate (positive or negative) of the falling letter. However, in our falling text we have not only letters but also blank movies. As collision detection is very costly, we use a condition statement. If it is a letter, the gate is closed and the image passes to the collision patch, otherwise the gate is open and we don't calculate collision detection. At the end we paste together the four columns of letters to come back to the initial size of the image and we send it to the parent patch.

3.7 Collision patch

Author : Philippe ADIB

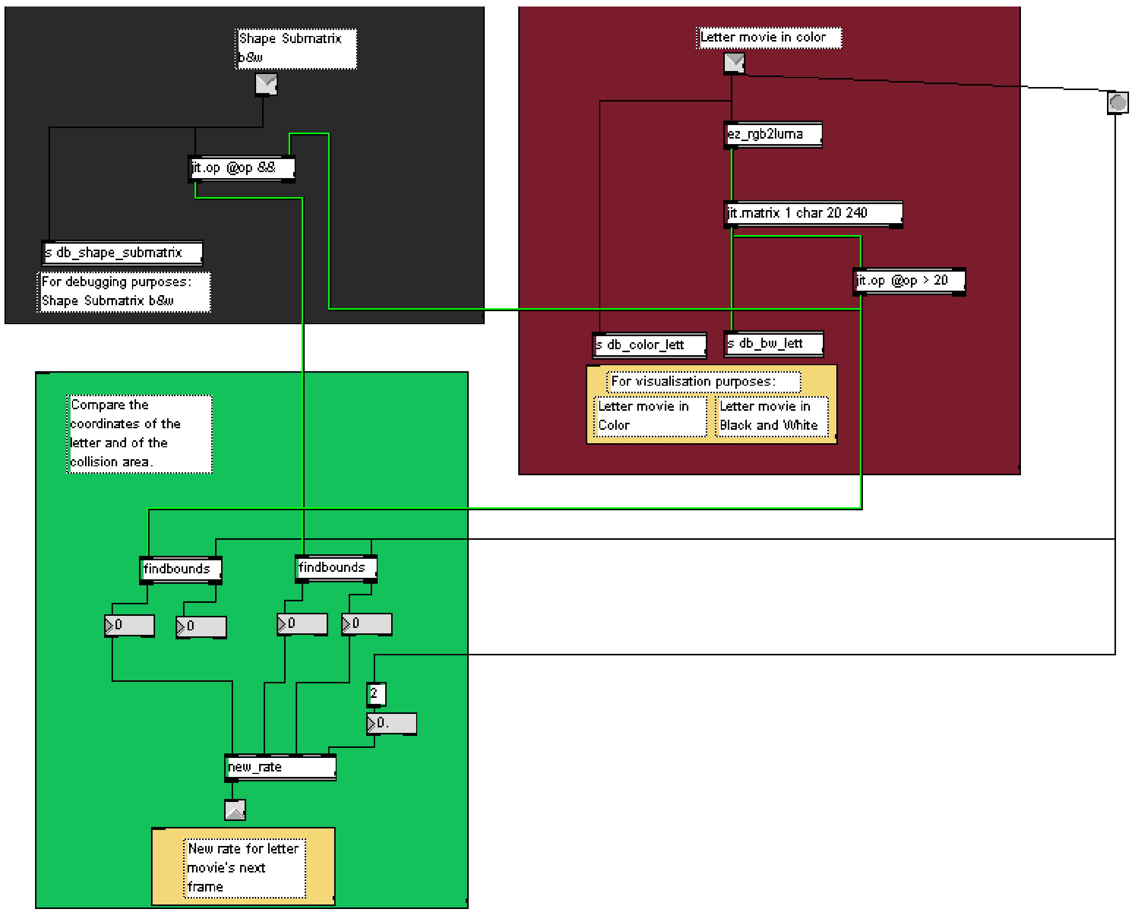


Figure 10: Collision patch

Collision detection was made using several Jitter objects: jit.op @op &&, jit.op @op > and jit.findbounds.

jit.op @op >

Feeding it as input a letter matrix consisting of a red letter on a black background, we set the argument of jit.op @op > to a value close to 0. What this did is that it took any value that was very close to black (ie≈0) and assigned it the value 0. For values not close to zero (ie > threshold) a value of 255 was assigned (since we're dealing with char matrices). Having set a value of 20 as threshold, the red and black video became black and white, where the red letter became a white letter on a black background. It would've been possible, yet much more strict, to use the jit.op @op != 0 object. This being so strict, could've caused certain very near 0 values to turn to white. By precaution, we did not follow that path.

jit.op @op &&

The jit.op @op && patch is an operator that can be used on two matrices. That is precisely what its use was in TextSnow. The two matrices in use were:

- A stripe of the shape video (in black and white), current frame.
- A letter video (in black and white), current frame.

Having constructed two black and white matrices of dimension 20x240, we can now perform a logical AND between them. This would result in a new matrix containing white values only where the white values of the original matrices coincide. As an illustration, take the following two matrices:

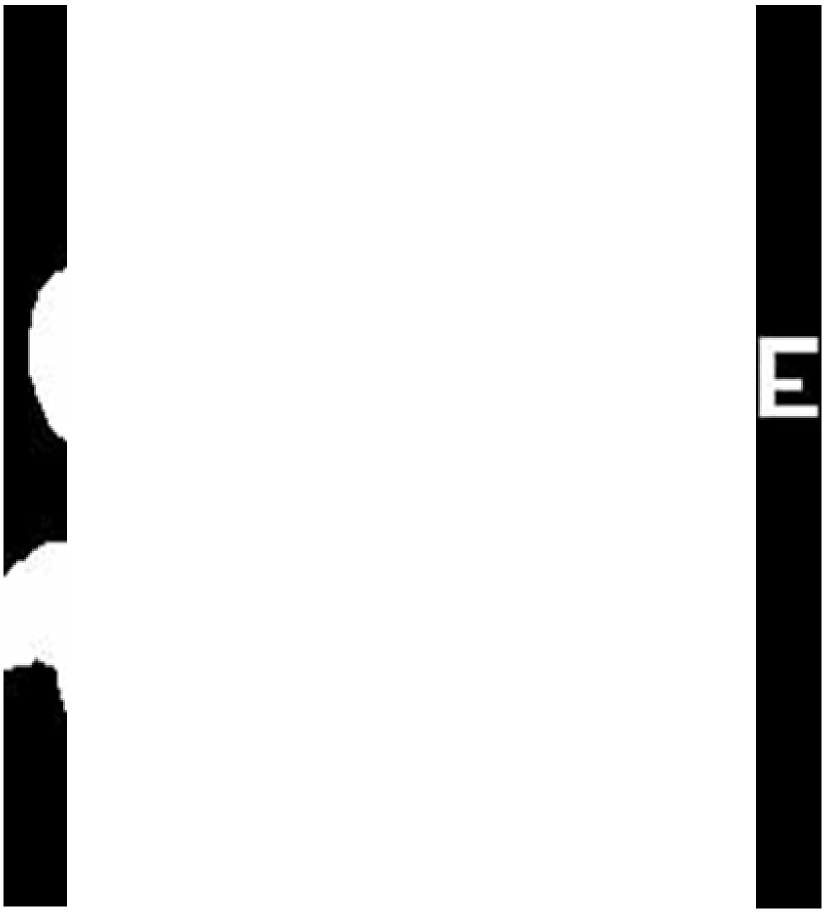


Figure 11: Shape and letter matrices

The resulting matrix of a logical AND performed on the above matrices would be their intersection, that is:

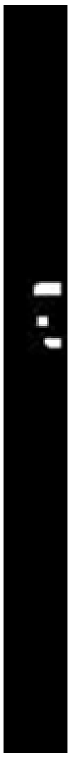


Figure 12: Intersection

As you can see, only part of the E in the letter matrix that coincided with the shape on the left exists in the final collision matrix. The way jit.op @op && works is that it does a cell to cell &&, whereby taking every cell of the LHS matrix and performing a logical AND with its corresponding equivalent in the RHS matrix. This equivalent cell is equivalent in terms of its position. In other words, a cell at row index 75 and column index 15 of the LHS matrix will have a logical AND performed on a cell at the same row and column indices of the RHS matrix. The result will be stored in a resulting matrix, each logical AND result stored at the same row and columns index of the cells on which the operation was originally performed.

jit.findbounds

Upon finding what we dubbed the "collision_matrix", the jit.findbounds patch was used in order to find the coordinates of the box encompassing all the white points of the collision matrix. What jit.findbounds does, is that it reads every cell of a matrix. But before doing so, it must initialize 4 values. Since jit.findbounds returns a box, this box must be defined by coordinates. The creators of jit.findbounds chose to return as a result the top left and bottom right corners of the encompassing box. For that reason, all four values of the two coordinates must be initialized. The value chosen for this purpose is -1, the reason for which being that, if no cell contains the a color in the specified range, then a value of -1 is returned for all coordinate values. Now, upon scanning the cells of the matrix, if it finds a cell containing a color within the accepted range, the coordinates of that cell are set as the minimum and maximum values for the encompassing box. As the scanning continues, the coordinates of the new scanned cells are used to update the coordinates of the encompassing box. In the case where one of the coordinate components of the cell under scan are smaller than the minimum coordinates or greater than the maximum coordinates, the corresponding component of the box's coordinates will be overwritten with the new value. In pseudo code, this algorithm would be written as:

```
if (Scanned_cell.x_coor < box.xmin) then box.xmin = Scanned_cell.x_coor;
if (Scanned_cell.y_coor < box.ymin) then box.ymin = Scanned_cell.y_coor;
```

and similarly, for the maximum coordinates of the box:

```
if (Scanned_cell.x_coor > box.xmax) then box.xmax = Scanned_cell.x_coor;
if (Scanned_cell.y_coor < box.ymax) then box.ymax = Scanned_cell.y_coor;
```

It is necessary that each component of the cell's coordinates be evaluated individually. Had the update required both components to satisfy a greater than or less than operation, many of the previous cases where only one component of a cell beats one of the box's in a less than or greater than fashion would be lost.

Finally, the encompassing box will not excessively contain all the cells whose color is within the specified range. This is translated in TextSnow in the following manner. The specified color range is any value near exactly white, that is 255,255,255 for the RGB scheme. Therefore, the resulting box will contain all the collision cells and some (black) background cells due to the square nature of the resulting box. The output coordinates of jit.findbounds will then be subsequently used in the "new_rate" patch, which is detailed in its own section in this document.

3.8 New_rate patch

Author : Pierre NOGUES

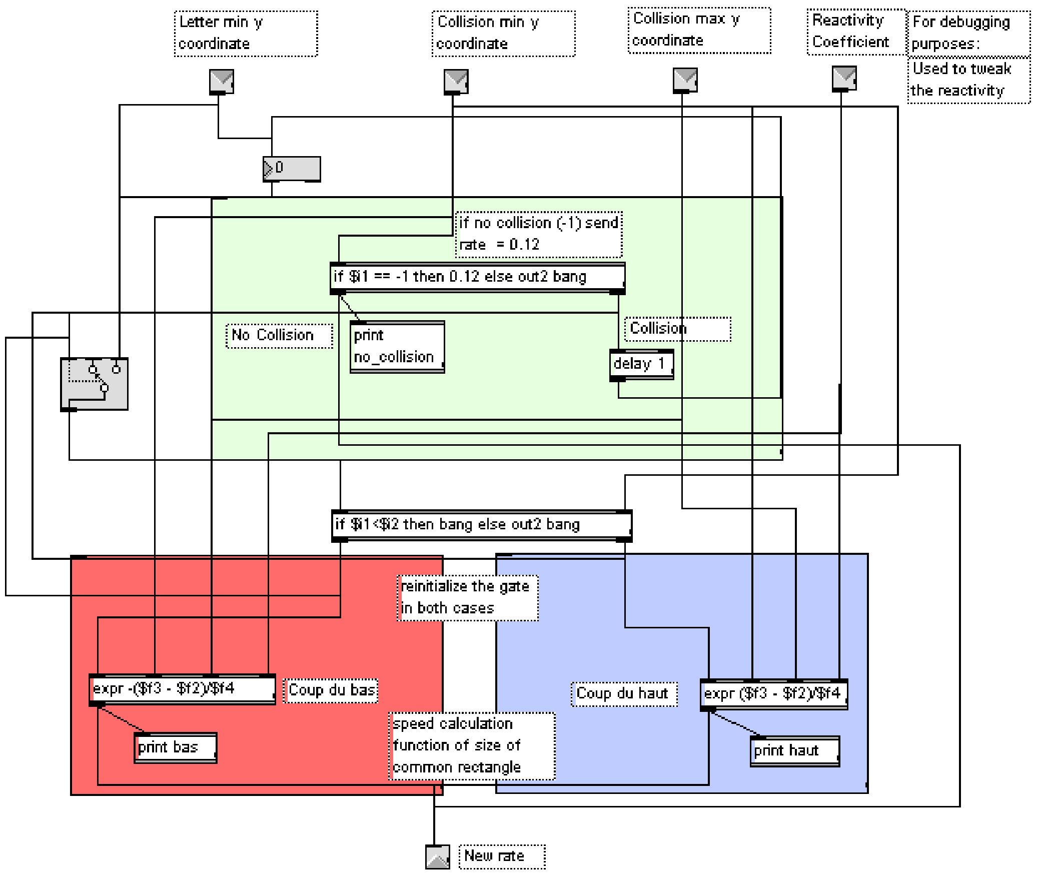

Letter min y coordinate
Collision min y coordinate
Collision max y coordinate
Reactivity Coefficient
For debugging purposes:
Used to tweak the reactivity
0
if no collision (-1) send rate = 0.12
if $i1 == -1 then 0.12 else out2 bang
No Collision
print no_collision
Collision
delay 1
if $i1<$i2 then bang else out2 bang
reinitialize the gate in both cases
expr -($f3 - $f2)/$f4
Coup du bas
print bas
Coup du haut
expr ($f3 - $f2)/$f4
print haut
speed calculation function of size of common rectangle
New rate

Figure 13: New rate

This patch calculates the new speed (rate) for playing the quicktime letter movie. A normal speed should be 0.12 (for a slow motion, like for snow). A negative value will make the movie playing backward.

In order to do that we need the bounding box of the letter and the bounding box of the part of the shape colliding with the letter. Then we determine how much these two boxes overlap each other. It is just algorithm, so there is no need of special Jitter objects. We use if/then statements to determine when a box overlap the other, and then if the result is positive we determine if the collision comes from the bottom of the letter or the top in order to move in the right direction the letter.

With Jitter, bounding boxes are manipulated using the upper left corner and lower right corner coordinates. Bounding boxes were calculated in the parent patch, collision.pat, using findbounds Jitter object as explained before. So in this patch, we handle the two Y coordinates to process the comparisons.

First we use Y coordinate of the collision bounding box to know if a collision occured or not. When -1 is received it means that no collision occured, otherwise a collision occured and we process the speed calculation. Then we compare the letter upper Y coordinate (which has a smaller value than the lower Y coordinate) and the shape upper Y coordinate to determine the side (top or bottom) of the knock. Then the speed is given by the formula Ymax - Ymin / coefficient, where Ymax - Ymin represents the height of the shape bounding box, so a unit in pixels, and coefficient is used to match this number with a reasonable speed. Finally the speed is positive for a collision coming from the top and negative for a collision coming from the bottom.

References

[1] Adam Lewensohnn, *Max Msp Jitter*
http://a.parsons.edu/~adam/fall05/max/index.htm

[2] CTIN 534 Experiments in Interactivity I, *Exploration and experimentation in design innovative interactive experiences*
http://interactive.usc.edu/members/534/archives/cat_maxmspjitter.html

[3] Tom Igoe: A Few Principles of Video Tracking
http://www.tigoe.net/pcomp/videoTrack.shtml

[4] Computer Vision for Artists and Designers: Pedagogic Tools and Techniques for Novice Programmers
http://www.flong.com/writings/texts/essay_cvad.html

[5] Unencumbered Full-Body Interaction
http://a.parsons.edu/~jonah/full_body/

[6] Cycling 74: Max/Msp/Jitter
http://www.cycling74.com

[7] Text Rain, *Camille Utterback & Romy Achituv, 1999*
http://www.camilleutterback.com

Demo link :

http://video.google.com/videoplay?docid=7518563856730771001&sourceid=docidfeed&hl=en-CA