Real Time Dynamic Face Warping Project

Kübilay SAVCI, Sarath Shekkizhar, Yi Zhang

May 10, 2013
Face Warping

Control Point Detection

Real-time face warping
Algorithm

Geometrical transformation of images is widely used in the world of digital image processing. For this project, we use the affine transform method among several different methods. Here, we have a base image which is transformed namely warped and then displayed on the screen as the output. The base image is placed in the ddr2 external memory on the DM6437 at board initialization. When the program starts up the marker recognition runs first. The markers are recognized and their locations are used as moving control points for the warping process. When the locations are obtained via captured frame their coordinates are injected into the warping function. Warping is concluded in a in hundreds of milliseconds and the result is displayed on the monitor. The whole process proved to be fast enough for a real time operation as we can ensure 10 fps display rate for the 720x480 YUV422 video format without any subsampling.

1.1 System Pseudo Code

Capture a frame from the camera

Scan the captured frame for markers

Warp the base image stored on the ext. mem.

Display the warped image

Missing markers

All Markers are detected
1.2 Face Warping

(a) Affine Transformation

Affine transform is used for geometrical transforms such as rotation, scaling, shearing and movement of images. The transform method preserves the quality of the image although fixed point operations result in minor artifacts due to precision loss. These issues will be addressed later on the challenges and optimization section.

Below, the forward mapping is illustrated which is used for constructing the destination image from the source image.

\[
\begin{align*}
F(p,q) & \xrightarrow{\text{TRANSFORM}} G(j,k) \\
F(u,v) & \xrightarrow{\text{image coordinate}} G(x,y) \\
\end{align*}
\]

F: Source image
G: Destination image

Pixelwise operations are done with the open form transform matrix below and it tells us where the pixel (the pixel at the location \((u,v)\)) should go on the output image space \((x,y)\).

\[
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\begin{bmatrix}
\cos \Theta & -\sin \Theta \\
\sin \Theta & \cos \Theta
\end{bmatrix}
\begin{bmatrix}
S_x & 0 \\
0 & S_y
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix} +
\begin{bmatrix}
m_x \\
m_y
\end{bmatrix}
\]

\(\Theta\): rotation angle

\(S_x, S_y\): Scaling Factors

\(m_x, m_y\): movement offsets
Since the forward mapping can result in unfilled holes in the output image, the reverse mapping is preferred in warping. The equation above defines the rules of transform per pixel. It can be generalized for triangular regions and can be written in a concise form as below.

\[
\begin{bmatrix}
  u_1 & u_2 & u_3 \\
  v_1 & v_2 & v_3
\end{bmatrix}
= \begin{bmatrix}
  b_1 & b_2 & b_3 \\
  a_1 & a_2 & a_3
\end{bmatrix}
\begin{bmatrix}
  1 & 1 & 1 \\
  y_1 & y_2 & y_3 \\
  x_1 & x_2 & x_3
\end{bmatrix}
\]

Above the coefficient matrix is the combined form of rotation-scaling-move matrices of the open form and it relates the output image to the source image so that it tells us where to pick up pixels from the source images while constructing the output. Using the above equation we can get to this equation below.

\[
\text{COEFF} = B \cdot A^{-1}
\]
(b) Triangles on the image

In order to initiate the transform, we form up fixed triangles on the source image and dynamically changing triangles using the marker positions. In this project we use 3 markers and we have 8 triangles in total as seen in the illustration below.

The source image is the head image which is stored in the external memory and it is never been overwritten. As fixed control points are changeable depending on the image, for the given image above we use the points defined in the code as below. Note that the origin is considered as the top-left corner of the image.

#define fx1 260 (left side of mouth)
#define fy1 320 (left side of mouth)
#define fx2 363 (midpoint of bottom lip)
#define fy2 335 (midpoint of bottom lip)
#define fx3 470 (right side of mouth)
#define fy3 329 (right side of mouth)
(c) Face Warping Pseudo Code

1. Construct the destination matrix (A) with obtained marker positions (x,y)
2. Inverse Destination Matrix ($A^{-1}$)
3. Find coefficients for each triangle using the source and inverse destination matrix
4. Triangle Testing & LUT
   - For every pixel on the output image, find in which triangle it lies
5. Pick up the corresponding coefficients depending on the result of triangle testing and do the reverse mapping (find u,v)
6. Bilinear Interpolation
   - Find the pixel value (YCbCr) from the source using mapping results (u,v)
7. Paste the calculated pixel value into the output image array
8. Display output image

From recognition

Pixel count < #pixels

Pixel count = #pixels

To recognition
(d) Barycentric Triangle Testing & LUT

In pixelwise operation for each pixel on the output we need to know in which triangle it lies, so that we can pick up the correct coefficient for the transform. To do so, we use a method called barycentric technique. In this method, the corners of the triangle and the point of interest on the image forms up a space. Assume that we have a triangle ABC and testpoints P and Q as shown in the figure.

The idea is that if we cross product \( u \times w \) and \( u \times v \), the output vectors are in the same direction whereas if we cross product \( u \times o \) and \( u \times v \), the vectors will be in the opposite direction. Using this information, we can figure out if the point is in the triangle or not.

As we have 8 triangles we are required to test a point for every triangle until we find the correct triangle. But this turns out to be cumbersome and takes long time especially if the pixel is in the last triangle it has to go through all triangle testing sequence. For this reason we came up with an idea such that in the first run we fill up a look table with the triangle information for each pixel and use it for next run. The LUT is updated after every warping process and it accelerates the process of finding the triangle since it is likely that a
pixel will be in the same triangle or in the neighboring triangles on the next warping run. The main reasoning is to change the sequence of triangle testing so that we can get to result much faster. Details are explained in the optimization section and more information can be found on the web regarding barycentric triangle testing.

**(e) Reverse Mapping and Bilinear Interpolation**

When we figure out the triangle namely the correct coefficient we can calculate the multiplication of the coefficient and destination matrices which gives us the mapping on the source for a given pixel on the output. As this operation implies it is reverse mapping and the coordinates \((u,v)\) we get may not be exactly integer coordinates. This means that the mapping may not land on an absolute pixel location namely \((u,v)\) can be fractional. If we select the closest pixel it creates distortions on the output image. Therefore we need interpolation.

In this project we use bilinear interpolation as it turns out to be efficient and practical in terms of speed.
If (x,y) above points out a location P between 4 pixels on the source image as seen in the figure, then the pixel value is the sum of the weights of the surrounding pixels. The weights are defined by the distance ratio to the corner pixels.

\[
f(x, y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1)\\ = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \left( f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1) \right).
\]
1.1 Marker Tracking

As explained in warping process the marker locations are needed in order to construct the destination matrix. The number of markers can be increased with the tradeoff of increasing processing time for warping hence we started with 3 yellow markers which are positioned around the mouth as seen in the figure.

Same number of triangles is formed up with the source image and since the markers are tracked the output image looks like it mimics the object in front of the camera.

Yellow color is chosen as the color for the markers because we think it is more salient compared to other colors.

Recognition is done via scanning a 20x20 window row by row and counting the number of yellow pixels in the window. If the sum of chroma values (Cb,Cr) is less than the threshold 220 than it is considered to be a yellow pixel. When the number of these pixels within a window exceed the threshold 20 it is considered to be a marker. The reasoning behind using a window is that due to change of illumination and camera characteristic some pixels can be displayed as yellow and this can result in false detections. Hence this window based thresholding eliminates all these kind of false
detections. Below in the figure, you can see that the yellow is detected and the pixels have been replaced with green for demonstration.

We should note that it can hit the same marker when the window moves to the next row while scanning and this can be regarded as another marker. To tackle this problem, we simply jump in X-direction by some offset and thereby skipping the previously detected marker. This will converge all detections for the same point into a single marker. After all markers are detected namely 3 markers then the function will pass it to warping process. But before than that, we sort the marker positions in order such that the leftmost marker is the 1st, the middle marker is the 2nd and the rightmost marker is the 3rd. As
Challenges

- Tracking Challenges
- Warping Challenges

1.1 Marker Tracking Challenges

Tracking posed several challenges which demanded a lot of tuning in the implementation to overcome these shortcomings. As mentioned earlier the tracking is done by sequential detection of Chroma (Cb, Cr) component based on a threshold followed by windowing over a 20x20 pixel window and counting the number of pixels that satisfy the “color threshold.” Now if the number of pixels that satisfy is greater than a certain percentage (“pixel threshold”) the position was taken as a possible marker position. The challenges faced before and on implementation of the above tracking, and the steps taken to overcome them are as listed below:

(a) Camera Perception

For a given color value the range of values corresponding to the color was far from the theoretical. These change from camera to camera and also on the lighting condition present. During initial implementation of the tracking threshold for color was tried for different values on a trial and error basis. Later a colored sheet was placed in front of camera and the range of values corresponding to the same color was checked and a threshold based on these observations was used.

(b) Environmental Conditions:

Although the pixels were given a range of color values for detection the range kept changing on different lighting conditions. Putting a larger color threshold affected the detection as other colors were
also detected. So our initial implementation of considering a given position to be a marker given all the pixels in an 8x8 window from the pixel proved less robust. Hence as mentioned an implementation where after a given pixel satisfies the color condition a larger window (20x20) was used and the number of pixels that satisfy the color condition within the window was used to detect a marker position based on a threshold. These thresholds were hardcoded after several tries and tweaking the value till a good and reasonable robustness was achieved.

Another problem that was faced was the detected position not being stable i.e unstable detection. Given a static marker due to the conditions that were presented the position of the marker was detected with different co-ordinates corresponding to it. This unstable detection was reduced by further changing the threshold values used but the values still were unstable. In order to overcome this problem, an implementation were warping was called only if the detected markers were at least some distance away (tried different distances and found 20 pixels distance to work) from the previously detected marker position. But this resulted in movements that were like discretized i.e the changes weren’t smooth. So a choice was to be made and so the above idea of stabilization using distance was dropped.

One more challenge that was faced was the outliers that got past the thresholding. This was overcome by keeping track of the number of markers detected and by calling the warping function only when the number was three. Also to avoid the same marker from contributing to detection as different markers an offset of 20 pixels was given for the next 20 rows following the detected marker position. This avoids redundant checking and detection.
(c) Speed of tracking:

One important thing all through the implementation of Tracking was to keep the speed of tracking much lower than the warping. As tracking should be able to run fast enough so that the sense of real time is kept up i.e even if it wasn't able to detect enough markers it should detect the markers in the next frame so that warping can be called and the output can be displayed. Instead of scanning every Frame fully the markers were searched for only in region of the frame were they can occur to provide for a possible speedup.

The speed of tracking also varies with number of outliers that are detected as once a pixel satisfies the color threshold it has to be processed with a window. The data provided below are at two different instances under different conditions of surrounding:

INSTANCE 1:

"21,827,720",1,"../main_demo.c", line 921: Function recognized called",Main Logger,

"26,275,015",2,"../main_demo.c", line 988: Warp call per frame",Main Logger,

"26,277,998",3,"../main_demo.c", line 192: Warp started",Main Logger,

Total time taken for warp to start: 4,450,278 ns

/////////////////////////////////////////////////////////////////////////////////////////////////////////

"115,584,276",6,"../main_demo.c", line 921: Function recognized called",Main Logger,

"118,687,207",7,"../main_demo.c", line 988: Warp call per frame",Main Logger,

"118,689,190",8,"../main_demo.c", line 192: Warp started",Main Logger,

Total time taken for warp to start: 3,104,914 ns

/////////////////////////////////////////////////////////////////////////////////////////////////////////

"212,735,255",11,"../main_demo.c", line 921: Function recognized called",Main Logger,

"217,128,446",12,"../main_demo.c", line 988: Warp call per frame",Main Logger,

"217,130,405",13,"../main_demo.c", line 192: Warp started",Main Logger,
Total time taken for warp to start: 4,395,150 ns

/////////////////////////////////////////////////////////////////////////////////////////////////////////
"308,760,892",16,"..main_demo.c", line 921: Function recognized called",Main Logger,
"313,233,627",17,"..main_demo.c", line 988: Warp call per frame",Main Logger,
"313,235,607",18,"..main_demo.c", line 192: Warp started",Main Logger,

Total time taken for warp to start: 4,474,715 ns

/////////////////////////////////////////////////////////////////////////////////////////////////////////
INSTANCE 2:
"250,765,568,097",1,"..main_demo.c", line 234: Function recognized called",Main Logger,
"250,771,367,740",2,"..main_demo.c", line 302: Warp started",Main Logger,

Total time taken for warp to start: 5,799,643 ns

"250,873,950,410",5,"..main_demo.c", line 234: Function recognized called",Main Logger,
"250,879,738,734",6,"..main_demo.c", line 302: Warp started",Main Logger,

Total time taken for warp to start: 5,788,324 ns

"250,971,090,346",9,"..main_demo.c", line 234: Function recognized called",Main Logger,
"250,976,865,811",10,"..main_demo.c", line 302: Warp started",Main Logger,

Total time taken for warp to start: 5,775,465 ns

"251,067,921,602",13,"..main_demo.c", line 234: Function recognized called",Main Logger,
"251,073,693,791",14,"..main_demo.c", line 302: Warp started",Main Logger,

Total time taken for warp to start: 5,772,189 ns
1.2 Warping Challenges

(a) Speed of warping

The major challenge that was faced with warping was making it run with a runtime that gives a reasonable FPS. Since warping formed the bulk of the time taken for the whole system, an improvement in warping in terms of speed resulted in a direct improvement in the FPS of the system. Initially the code was implemented using floating point operations and this was a major shortcoming and resulted in slow processing as the board is of the fixed point type. Several optimizations were done on the code till the time taken was reduced considerably to provide for real time behavior.

(b) Integer Overflows

One other challenge that was faced as a result of converting floating points to fixed points was integer overflows. This occurred in triangle testing method and while calculation of inverses for certain triangles. This was overcome by scaling down the terms used for processing that caused an overflow in the triangle test process. The scale down was done by using right shift operator. While conversion for inverse calculation the triangles that had overflows was done in floating point but the co-efficient obtained as a result were stored as integers. The calculation of inverses for these triangles was done once for a Frame and so floating operations didn’t cost as much as those that are done per pixel.

One instance were Overflow occurred in triangle testing is shown below with values that were obtained as a result of overflow and as a result of scaling. The scaling was done on the dot product results by right shifting each result by 3 before using it for other purposes. The overflow occurs in variables ‘u’, ‘v’ and ‘denominator’ which are
obtained using the dot variables. As a result of scaling the overflow variables are scaled down by 64 (i.e right shift by 6). Also notice the precision error introduced in variable ‘u’.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Original value (float)</th>
<th>Overflow values (int)</th>
<th>Scaled values (int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>denominator</td>
<td>4.6526489e+010</td>
<td>-718150256</td>
<td>726975416</td>
</tr>
<tr>
<td>dot00</td>
<td>152500.00</td>
<td>152500</td>
<td>19062</td>
</tr>
<tr>
<td>dot01</td>
<td>179750.00</td>
<td>179750</td>
<td>22468</td>
</tr>
<tr>
<td>dot02</td>
<td>35000.0000</td>
<td>35000</td>
<td>4375</td>
</tr>
<tr>
<td>dot11</td>
<td>516960.0000</td>
<td>516961</td>
<td>64620</td>
</tr>
<tr>
<td>dot12</td>
<td>100660.00</td>
<td>100660</td>
<td>12582</td>
</tr>
<tr>
<td>u</td>
<td>0.00000000</td>
<td>0</td>
<td>20124</td>
</tr>
<tr>
<td>v</td>
<td>9.0593997e+009</td>
<td>469465408</td>
<td>141540584</td>
</tr>
</tbody>
</table>

(c) Precision Error

One more problem that arose as a result of converting to integer operation was the error in triangle testing as the pixels along the edges of the triangle were no more accurately described by triangle testing and fell out of all triangles. This resulted in erroneous outputs displayed as Noise on the output. This was overcome by assuming the previous pixel mapping for those pixels that didn’t satisfy any triangle testing method. This was reasonable assumption as given the image is processed sequentially row by row; the chances of the pixel mapping for adjacent pixel will be close. This assumption was also used in speeding up the warping algorithm – Pixel Localization. Thus by localizing the pixels the errors that resulted due to precision were solved.

Precision also mattered when scaling co-efficient for mapping. As mentioned above scaling up was done using left shift operator and shifting by 7 (approximately multiplying by 100) resulted in visible mapping changes in adjacent triangles i.e there was a visible difference in the output along the edges of the triangles constructed on the image.
Optimizations

Several optimizations were done on the implementation of warping, of which the following are those that made big impact on the runtime of the algorithm.

- FIXED POINT PROCESSING
- LOOK UP TABLE
- PIXEL LOCALIZATION
- AREA OF INTEREST
- INLINE FUNCTIONS

The initial runtime of the algorithm before all these optimizations and the time taken per pixel are as listed below:

31958315686,1,"../main_demo.c", line 395: Warp started" ,Main Logger,
31958324781,2,"../main_demo.c", line 444: Transform Calculation" ,Main Logger,
31958390730,3,"../main_demo.c", line 462: Triangle 1 done" ,Main Logger,
31958425942,4,"../main_demo.c", line 478: Triangle 2 done" ,Main Logger,
Time taken for finding co-efficient for all triangles: 316,353 ns

Time taken for one warp function process: 19,287,902,196 ns

Time for mapping one pixel: 68,101 ns

Time taken by a single Barycentric testing: 32,693 ns

Time between Calculate input call and Barycentric call: 6,776 ns

Also notice in the above case the triangle testing was satisfied with the first test itself, but it is not necessary that every pixel will take above time. In fact the above value will be the lower bound for time
taken for each pixel and in worst case the triangle testing runs eight times increasing time resulting in 296,952 ns.

One more observation that was made on runtime analysis is the fact by nested calling functions like one seen above where “Calculate input” calls “Check Points” which in turn calls “Barycentric” some time is taken with every call though no processing is done. This may very well be the time taken to push variables onto stack and retrieving them. Though this takes only 4,715 ns in case of check points called by calculate input function, it proves valuable when it is done for every pixel. The time taken by nested function calls depends on the number of variables defined in the function calling.

1.1 Fixed Point Processing:

Converting to fixed - point processing produced a much improved runtime reducing the time taken by several factors.

a) Converting Triangle testing to integer operation with downscaling to avoid overflows. Also the nested function calls like the ones within “Barycentric” testing and “Check point” function were made inline.

"6,373,350",1,"../main_demo.c", line 367: Warp started",Main Logger,

"6,709,867",2,"../main_demo.c", line 546: generation of images started",Main Logger,

"6,712,508",3,"../main_demo.c", line 181: Calculate input started",Main Logger,

"6,714,930",4,"../main_demo.c", line 155: Barycentric_int started",Main Logger,

"6,719,629",5,"../main_demo.c", line 173: Barycentric_int done",Main Logger,

Time taken by a single Barycentric testing: 4,699 ns

Time between Calculate input call and Barycentric call: 2,422 ns

Notice that by just converting from floating point to fixed point the time taken by triangle testing has been reduced by a factor of
approximately 7. Also notice that removing nested function reduces time by a factor of 3.

As a result of the above optimization the total time taken by the warping function was reduced considerably. The “Calculate input” function was also made inline to reduce time as much as possible. The results are as below.

"71,583,386,833",1, ""../main_demo.c", line 255: Warp started",Main Logger
"71,583,760,171",2, ""../main_demo.c", line 441: generation of images started",Main Logger
"78,521,801,797",3, ""../main_demo.c", line 581: Warp done",Main Logger

Time taken for one warp function process: 6,938,414,964 ns

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

"78,530,802,762",4, ""../main_demo.c", line 255: Warp started",Main Logger
"78,531,161,245",5, ""../main_demo.c", line 441: generation of images started",Main Logger
"85,469,352,750",6, ""../main_demo.c", line 581: Warp done",Main Logger

Time taken for one warp function process: 6,938,549,988 ns

The above is approximately an improvement of runtime by a factor of 3.

b) The optimization that was done following the above was basically a down sample by 2 in processing by taking the value adjacent to a given pixel to have a value same as that of its previous pixel in the row. So while storing value in YCbCr format for the display instead of calculating the luma component for two pixels the same value was copied to both pixels. The resulting runtime was as expected reduced by a factor of 2 as in results below.

"14,266,356",1, ""../main_demo.c", line 349: Warp started",Main Logger
"14,645,803",2, ""../main_demo.c", line 528: generation of images started",Main Logger
"2,627,418,853",3, ""../main_demo.c", line 552: Warp done",Main Logger

Time taken for one warp function process: 2,613,152,497 ns
c) Now all coefficients were made integer and since we can’t just lose precision in the coefficients, an up scaling factor basically shifting left by 10 bits was used before processing. And certain coefficients corresponding to a given triangle were redundant and hence removed. For instance in Triangle 1 as defined earlier in algorithms the coeff[0][0] was always 1. Also as discussed earlier some coefficients overflow when converted to integer by scaling as above and so these were left floating point while calculation but converted integers finally before being used for mapping every pixel.

The runtime reduced considerably again and the results are as below.

Time taken per one warp function process: 177,394,697 ns

1.2 Look up Table

One other optimization was based on using inter-frame dependency to the advantage of warping. The reasoning was a given pixel belonged to a particular triangle the chances of it belonging to the same triangle or the nearby ones is much higher than some triangle which is impossible to land upon. So the sequence of triangle testing was changed based on where the pixel
was located in the previous frame and update in every frame. This gave reasonable improvement in runtime.

"20,647,418,329",1,"../main_demo.c", line 242: Warp started",Main Logger,
"20,647,807,170",2,"../main_demo.c", line 436: generation of images started",Main Logger,
"20,905,265,624",3,"../main_demo.c", line 824: Warp done",Main Logger,

Time taken for First warp function process: 257,458,454 ns

///////////////////////////////////////////////////////////////////////////////////////////////////

"20,647,418,329",1,"../main_demo.c", line 242: Warp started",Main Logger,
"20,647,807,170",2,"../main_demo.c", line 436: generation of images started",Main Logger,
"20,905,265,624",3,"../main_demo.c", line 824: Warp done",Main Logger,

Time taken for Second warp function process: 152,623,717 ns

///////////////////////////////////////////////////////////////////////////////////////////////////

"21,008,491,215",4,"../main_demo.c", line 242: Warp started",Main Logger,
"21,008,868,180",5,"../main_demo.c", line 436: generation of images started",Main Logger,
"21,161,491,897",6,"../main_demo.c", line 824: Warp done",Main Logger,

Time taken for Third warp function process: 119,873,088 ns

"21,008,491,215",4,"../main_demo.c", line 242: Warp started",Main Logger,
"21,008,868,180",5,"../main_demo.c", line 436: generation of images started",Main Logger,
"21,161,491,897",6,"../main_demo.c", line 824: Warp done",Main Logger,

Time taken for Fourth warp function process: 122,911,802 ns

///////////////////////////////////////////////////////////////////////////////////////////////////

"21,714,654,993",13,"../main_demo.c", line 242: Warp started",Main Logger,
"21,715,031,523",14,"../main_demo.c", line 436: generation of images started",Main Logger,
"21,841,475,735",15,"../main_demo.c", line 824: Warp done",Main Logger,

Time taken for Fifth warp function process: 126,444,212 ns

It was observed that certain coefficients corresponding to a given triangle were calculated redundantly and hence the redundancies
were removed. For instance in Triangle 1 as defined earlier in algorithms the coeff[0][0] was always 1. Also as discussed earlier some coefficients overflow when converted to integer by scaling as above and so these were left floating point while calculation but converted integers finally before being used for mapping every pixel.

The results obtained by removing certain inverse calculation are as below.

"7,706,216,934",2,"../main_demo.c", line 175: Warp call per frame",Main Logger,
"7,706,218,973",3,"../main_demo.c", line 280: Warp started",Main Logger,
"7,706,480,282",4,"../main_demo.c", line 435: generation of images started",
"7,842,335,262",5,"../main_demo.c", line 823: Warp done",Main Logger,

**Time taken for First Frame: 135,854,980 ns**

"7,859,256,138",6,"../main_demo.c", line 110: Function recognized called",
"7,874,842,994",7,"../main_demo.c", line 175: Warp call per frame",Main Logger,
"7,874,845,084",8,"../main_demo.c", line 280: Warp started",Main Logger,
"7,875,093,505",9,"../main_demo.c", line 435: generation of images started",
"7,996,304,565",10,"../main_demo.c", line 823: Warp done",Main Logger,

**Time taken for Second Frame: 121,211,060 ns**

"8,013,321,803",11,"../main_demo.c", line 110: Function recognized called",
"8,029,549,297",12,"../main_demo.c", line 175: Warp call per frame",Main Logger,
"8,029,551,491",13,"../main_demo.c", line 280: Warp started",Main Logger,
"8,029,823,848",14,"../main_demo.c", line 435: generation of images started",
"8,151,051,535",15,"../main_demo.c", line 823: Warp done",Main Logger,

**Time taken for Third Frame: 121,227,687 ns**
1.3 Pixel Localization

Now given a pixel in a triangle the chance of the pixel in the following row and same column belonging to the same triangle is very high and so a triangle testing for these pixels seems redundant. Hence instead of testing for which triangle the neighbor pixel belongs, it was taken to be in the same triangle as the one in the previous row. This reduced the time taken by warping and hence the overall time of the system.

"250,765,568,097","../main_demo.c", line 234: Function recognized called",Main Logger,

"250,771,367,740","../main_demo.c", line 302: Warp started",Main Logger,

"250,771,579,069","../main_demo.c", line 444: generation of images started",Main Logger,

"250,862,512,754","../main_demo.c", line 987: Warp done",Main Logger,

Time taken for First Frame: 96,944,657 ns

Time taken for First warp function process: 91,145,014 ns

////////////////////////////////////////////////////////////////////////

"250,873,950,410","../main_demo.c", line 234: Function recognized called",Main Logger,

"250,879,738,734","../main_demo.c", line 302: Warp started",Main Logger,

"250,879,944,139","../main_demo.c", line 444: generation of images started",Main Logger,

"250,959,649,309","../main_demo.c", line 987: Warp done",Main Logger,

Time taken for Second Frame: 85,698,899 ns

Time taken for Second warp function process: 79,910,575ns

////////////////////////////////////////////////////////////////////////


Further Improvements

1.1 Marker Tracking

(a) More sensitive marker detection

In the further modification, we are planning to improve the performance of marker tracking, which is to make this detection more sensitive. In other words, no matter how big expression users are making and no matter what the background environment is, we intend to keep the detection efficiently enough.

(b) Diverse markers

Include the markers stick on user`s face and magnetic stickers, we are planning develop more convenient and efficient way to get the major movement of user`s expression change.

1.2 Face Warping

(a) Improve the processing speed

Even though we are now achieve the processing speed less than 100ms,we intend to further reduce the processing time by upgrading the warping algorithm.

(b) Setting up more control points

To show the expression change more clearly and vivid, we may also add more control points and further divides the face into more triangles.
Applications

The system could be added to video chat clients as a possible effect where the user instead of sending his video feed can send another person talking. This would be a cool and fun effect that would make chats more interesting. To be used in such an application the system needs to handle different resolution and more importantly a **marker free detection** of control point. One more additional thing that needs to be considered is the orientation of the face itself. The system needs to detect when the user is captured in different focus and angles by the camera and use a corresponding model to process. This will not be a problem as memory issues of the DSP will no longer be a limiting condition when the system is marketed.

The system if designed in such a way could be marketed as **software plug-in** in video chat software like Skype or Goggle hangout or as part of the video clients itself. There is no additional cost required for using such a plug-in as it requires no additional hardware. The plug-in can be marketed for various user-ends namely from mobiles, tablets to laptops, desktops.

Also one can view the system to give way for better face graphics in animation models for both the gaming industry and for the movie industry. But here we don’t need the real time sense and so we can take advantage of this fact and save input frames and then process and render additional features if necessary.
References

[1] Point in triangle testing
http://www.blackpawn.com/texts/pointinpoly/
