Alignment invariant image comparison implemented on the GPU

Hans Roos
Highquest, Johannesburg
hans.jmroos@gmail.com

Yuko Roodt
Highquest, Johannesburg
yuko@highquest.co.za

Willem A. Clarke, MIEEE, SAIEE
University of Johannesburg
willemc@uj.ac.za

Abstract

This paper proposes a GPU implemented algorithm to determine the differences between two binary images using Distance Transformations. These differences are invariant to slight rotation and offsets, making the technique ideal for comparisons between images that are not perfectly aligned. The parallel processing capabilities of the GPU allows for faster implementation than on traditional desktop processors. In order to take full advantage of this all aspects of the algorithm was implemented on the GPU.

Key words: Distance transform, binary image, GPU, parallel processing.

1. Introduction

In the field of image processing, image comparison has a wide variety of applications. These applications range from image retrieval to image registration [1]. In this paper we are proposing to make use of graphics processing units (GPU), parallel processing techniques and distance transformations to compare images invariant to slight rotation or offsets.

The GPU was selected for this purpose due to its computational power. Recent advances in graphics architecture have ensured that GPUs have extensive memory bandwidth along with tremendous increases in its computational horsepower. These increases are clearly advantageous. Other advantages of GPU algorithm implementations include the fact that GPUs can perform these operations faster and their cost versus computational power is much lower than that of central processing units (CPU) [7, 8]. GPUs also provide better performance per thread than CPUs can provide [7]. The mentioned advantages have given GPUs a popular position amongst researchers to use them for general purpose computations [8, 9]. GPUs do however have their own set of disadvantages: "they lack some fundamental computing constructs" [8]. The absence of these constructs make GPUs ill suited for tasks such as cryptography.

The Distance Transformation (DT) is an operation performed on binary images (images containing black and white pixels; or feature and non-feature pixels) which returns a greyscale representation where each pixel value represents *that* co-ordinate's distance from its nearest feature pixel in the binary image [3, 9]. The Distance Transform is an important tool in image processing; however its uses have extended into other fields including that of pattern recognition computer vision, computer graphics to name a few [4, 9].

Various methods of determining Distance Transformation exist. In this paper we utilize the 4-connected distance (otherwise known as the city block distance map) [6]. Other distance maps such as the Euclidean distance map may also be used. The Euclidean map is described as a map which corresponds to how real world objects are measured, which makes it easily interpreted. That said, the brute force approach to calculating the Euclidian distance is not feasible as it involves measuring the distance between every feature pixel and every non-feature pixel yielding a computational complexity of $O(n^2)$ for every pixel [11]. However the 4-connected approach is the least complex and provides a good enough approximation of the distance for the purpose of this application.

2. Definitions

In this section we will more clearly define the concepts of binary images and distance transformations. These definitions are to be used at a later stage.

A point on an image can be defined in terms of x and y such that $x \in \{1,...,width\}$ and $y \in \{1,...,height\}$, where width and height are the dimensions of the image. Hence (x, y) is an arbitrary point on the image.

Adding to the earlier definition of a binary image it can be stated that binary images contain foreground pixels and background pixels. The foreground pixels represent the objects in the image. Thus it can be written as follows:

A binary image can be represented as a function, $I(x, y)$ where $I(x, y) \in \{O, B\}$. O and B represents object and background pixels respectively; in terms of implementation $I(x, y) \in \{1,0\}$. In other words the notation states that the texture value at the point *(x, y)* is either a foreground pixel or a background pixel.

For the definition of the Distance Transform, we can say: the Distance Transform can be represented by the function, $D(x, y)$ where $D(x, y) \in \{0,...,1\}$. The set $\{0,...,1\}$ is the distance to the nearest foreground pixel, the range of this set may vary depending on implementation, convention and preference. For example $D(x, y) \in \{1,...,imagesize\}$

In this paper we will refer to the input image and the image to be compared, as $I_1(x, y)$ and $I_2(x, y)$ respectively. For each comparison two Distance Transformations are required, one for all the distances to the nearest object, $D_O(x, y)$ and the other all the distances to the nearest background pixel $D_B(x, y)$. These

transformations are only done for one of the input images; however both are done on the same input image.

3. Implementation

In this section we will discuss the implementations of the main components of the papers, namely the Distance Transformations and then the comparison algorithm.

3.1 Distance Transform Implementation

Initially our distance map was approximated using the concept of a local distance map. The distances were calculated around each pixel, but only for a small region or window as implemented by É. Baudier et al using the Hausdorff distance [1]. However, our implementation used a circular window around each point and the Euclidian distance between each pixel in the window and the centre of the window.

The 4-connected distance transform is implemented by selecting the minimum value between a pixel's four surrounding values (above, below, left and right) and storing them into an interim distance map. This interim distance map is then passed back and is recursively processed until all the distances have been computed [6].

3.2 Example of the distance transform

For the purpose of clarity the colours of the images have been inverted, i.e. black represents the foreground and white represents the background as opposed to the norm where white represents the foreground (features) and black the background (non-features). Figure 1 shows a binary image containing two objects, where figure 2 represents figure 1's distance transformation. In figure 2, the darker the colour, the further away from the object the point is.

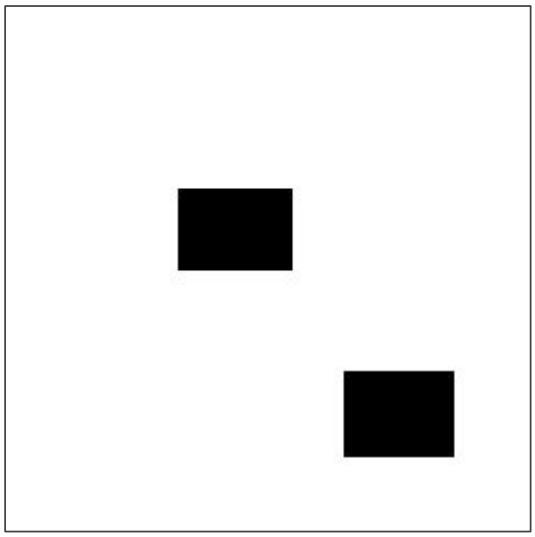


Figure 1. Binary image containing 2 objects represented as black pixels.

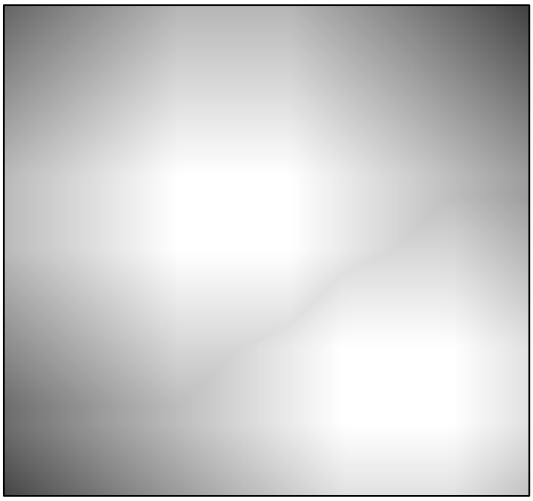


Figure 2. Figure 1's Distance Transform. Darker colours are far from the objects.

3.3 Image Comparison Implementation

The proposed algorithm is as follows: in order to compare the two images the distance maps of the first image, $I_1(x, y)$, have to be computed; with respect to both background pixels and foreground pixels i.e. two distance maps are created. One containing distances to the nearest white feature, $D_O(x, y)$ and another containing distances to the nearest black feature, $D_B(x, y)$.

Once these distance maps have been acquired a pixel at point (x, y) from the second image, $I_2(x, y)$, is compared to the 2 distance maps. If the pixel at the *current* point is black the corresponding distance value in the nearest-to-black map, $D_B(x, y)$, is returned. If the pixel is white the corresponding value is returned from the nearest-to-white map, $D_O(x, y)$. The output of the algorithm then represents the differences in the image, or rather how far a point is to its closest feature. Figure 3 shows a graphical representation of the algorithm where D_O, D_B and I_2 are the input textures.

In terms of the GPU implementation of the algorithm; OpenGL fragment programs were coded to generate the two distance maps of the first image, I_1. The distance maps are stored in the GPU's memory as a texture (or image). This is done to avoid losing the GPU's performance advantage by passing information back and forth between the GPU and CPU. The second distance map is done using the same algorithm as the first. However, the inverse of the first image is used as an input. The inversion is also implemented on the GPU. A separate fragment program was created in order to do the comparison on the GPU. The result of the comparison is then stored as a texture and then displayed on screen.

The pseudo code below is the algorithm for comparing the 2 images as implemented in the comparison fragment program. The value `current_Pixel` is the current texture value from the second image, I_2.The value `current_Distance` is the texture value from either one of the two distance maps at the

current *(x, y)* position; the same position where `current_Pixel` was obtained. The `current_Distance` is returned to a new texture in order to make the result graphically viewable.

```
current_Pixel<- current_Texture from I2

if current_Pixel = black then
   current_Distance<- value from DB
else
   current_Distance<- value from DO

return current_Distance
```

From the algorithm it is easy to see that a threshold can be added which can be used to make decisions based on the result, for example to discard any differences that are not intense enough and only keep the differences that are clear enough.

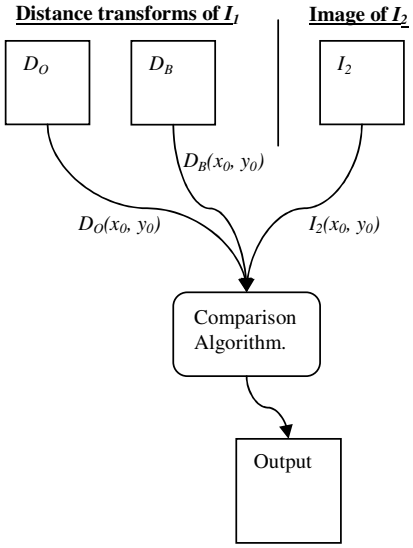


Figure 3. Graphical representation of the algorithm

4. Experimental Setup

The algorithm was tested on two different systems; both systems had Windows XP Professional 32Bit Service Pack 3 as operating systems. The main specifications of the two systems are as follows:

	System A	System B
CPU	AMD Athlon X2 4200+	AMD Athlon 3200+
GPU	8800GTX	6800GE
RAM	2048 MB	2048 MB

Table 1. System used in the performance test of the algorithm

The systems were chosen as they are from two different eras in terms of performance, System A being a lot more powerful than System B especially in terms of graphics processing capabilities.

	8800GTX	6800
Pixel Shaders	128	16
Core Clock (MHz)	575	350
Memory (MB)	768	256
Memory Clock (MHz)	900 (DDR3)	500 (DDR3)
Shader Model	4.0	3.0

Table 2. GPU specifications of the test systems

The algorithm was initially written and implemented in RenderMonkey (version 1.81) to test and verify the OpenGL syntax. Once verified, the OpenGL was implemented in C++ in order to do more accurate performance tests and comparisons between the two systems.

5. Results

The results of the tests will be discussed in the following section. Firstly we will look at the results of the image comparisons followed by the performance results

5.1 Comparison Results

The algorithm was tested on various images. One of the tests was done on a "spot the difference" game containing eight differences. The results are discussed below.

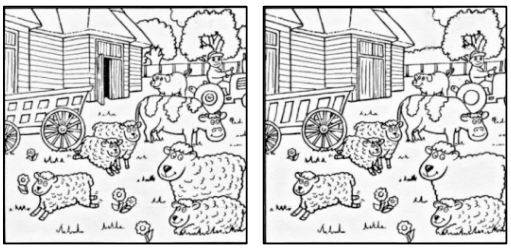


Figure 4. Input images. Spot the difference game containing 8 differences [10].

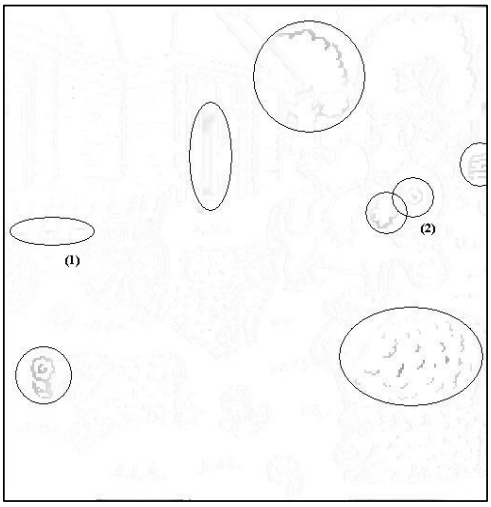


Figure 5. Highlighted differences between the images in figure 4.

Figure 5 (above) highlights the differences between the images in figure 4. Circles were placed round all eight of the differences. The comparison seems to fail in regions where it is difficult to compute distances as the differences are only subtle changes in shape, see points 1 and 2 on figure 5. The fact that these appear as light grey, shows that the algorithm is only recognizing a minor difference. The grey outlines of the images above are due to the fact that the images are not perfectly aligned for demonstration purposes; showing the invariance property of the algorithm.

Further tests were done with regards to more practical applications such as template matching and character recognition. Figure 6 a and b (the numbers "3" and "8") were compared. The result of the comparison can be seen in figure 7.

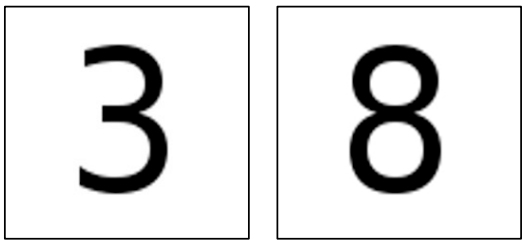


Figure 6 a and b. The second part of the algorithm test demonstrating possible uses in template matching and character recognition.

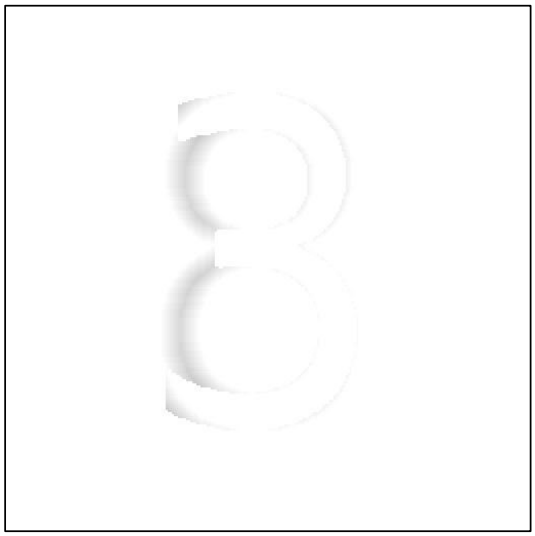


Figure 7. Comparison results between figure 6 a and b

When comparing the image (example figure 6 a) to a slightly rotated version of itself (figure 8 a) using the proposed technique, only minor differences are highlighted (see figure 8 b). These changes can easily be discarded. However, when comparing our results to an XOR comparison, the rotation is clearly visible in the output (see figure 8 c). Rotating the image further, still only highlights minor changes when using our technique. Again the XOR comparison reveals very clear changes due to the rotation (see figure 8 d, e and f).

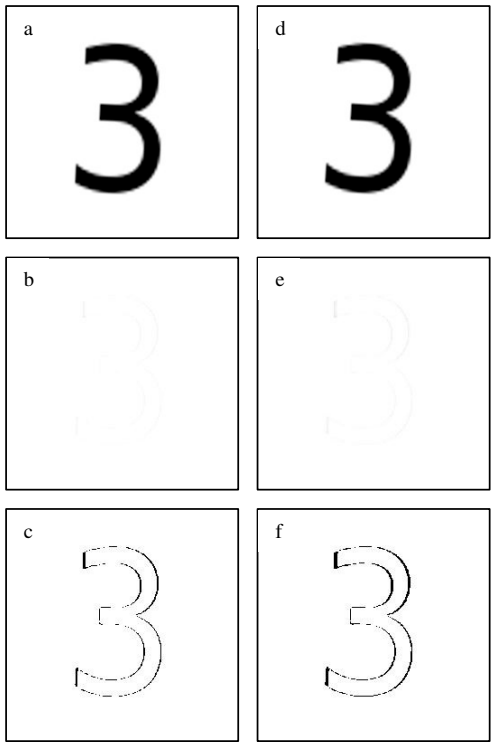


Figure 8 a – f. Comparison of images with rotation using the proposed algorithm and XOR comparisons.

Figure 9 demonstrates a situation where the comparison will begin to fail. The rotation of the image is much greater than the previous examples. However it will still be possible to threshold out and discard many of the errors, but in such an extreme case it leaves a lot of room for error.

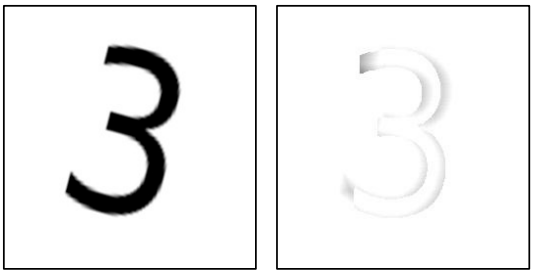
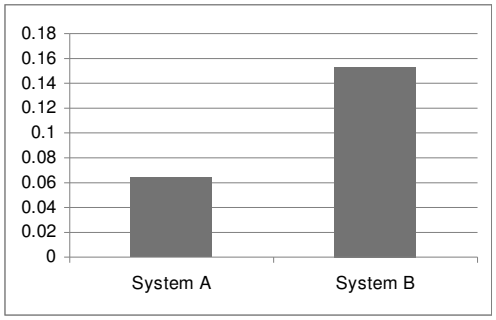


Figure 9. Comparison demonstrating a larger rotation, where inevitably the test will begin to fail.

5.2 Performance test results

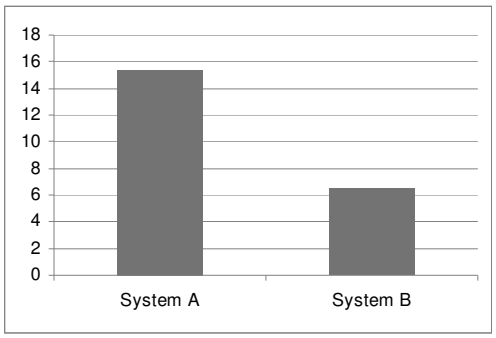
The performance was tested in terms of processing time and frames per second (fps). When we refer to frames per second we are referring to actual renders per second which is the inverse of the processing time. The tests on both systems were done using 128 iterations to calculate each Distance Transformations. Images of size 1024 by 1024 were used in the tests. The performance results have been summarized as follows.

Processing time (s)



Graph 1 Processing time (s)

Frames per second (fps)



Graph 2 Frames per second (fps)

The above results show that the performance of the algorithm is sufficient even on older systems. Considering the fact that number of calculations done to perform just one of the Distance Transformations is a staggering 134,217,728 iterations ($1024 \times 1024 \times 128 = 134217728$).

6. Conclusion

We proposed a technique to comparing images using the concept of distance maps. The entire algorithm was implemented on the GPU in OpenGL to take maximum advantage of the performance advantage of the GPU has over traditional desktop processors.

The comparisons were invariant to slight rotation and offset as seen in the comparison results. This invariance makes the technique useful in the fields of template matching and character recognition.

7. References

[1.] Baudrier É, Nicoler F, Millon G and Ruan S, "Binary-image comparison with local dissimilarity mapping", Pattern Recognition, Vol. 41, pp. 1461-1478, 2008

[2.] Kim HY and Araújo SA, "Grayscale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast," IEEE Pacific-Rim Symposium on Image and Video Technology, Lecture Notes in Computer Science, vol. 4872, pp. 100-113, 2007.

[3.] Gavrila DM, "Multi-Feature Hierarchical Template Matching Using Distance Transforms", Proceedings of 14th International Conference on Pattern Recognition (ICPR'98), Vol. 1, pp 439, 1998

[4.] Felzenszwalb PF, Huttenlocher DP, "Distance Transforms of Sampled Functions", Cornell, Computing and Information Science, 2004

[5.] Saude AV, Couprie M, De Alencar Lotufo R, "Distance Transform to seeds: computation and application", Proceedings of 8th International Symposium on Mathematical Morphology, Vol. 2, pp 15-16, 2007

[6.] Bailey DG, "An Efficient Euclidean Distance Transform", International conference on combinatorial image analysis, IWCIA, Vol. 3322, pp. 394-408, 2004

[7.] Owens J, Davis UC, "GPU Architecture Overview" Proceedings of International Conference on Computer Graphics and Interactive Techniques, Article 2, 2007

[8.] Owens J, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn A and Purcell T. "A Survey of General-Purpose Computation on Graphics Hardware". Proceedings in Eurographics, State of the Art Reports, pp. 21-51, 2005

[9.] Rong G, Tiow-Seng T, "Jump Flooding in GPU with Applications to Voronoi Diagrams and Distance Transform", Proceedings in ACM Symposium on Interactive 3D Graphics and Games, pp. 109-116, 2006

[10.] "Wimphole Home Farm", "spot the difference image" found at http://www.wimpole.org/spot.html, 25 September 2008

[11.] Fabbri R, JC Torelli, Bruno OM, "2D Euclidean Distance Transform Algorithms: A Comparative Survey", ACM Computing Surveys, Vol. 40, No. 1, Article 2, 2008.