UTILIZING SPATIAL LOCALITY OF COLOURFAST FEATURES FOR GPU-ACCELERATED OBJECT RECOGNITION

A thesis submitted to Auckland University of Technology in partial fulfillment of the requirements for the degree of Master of Science
April, 2015

Supervisors
Dr Andrew Ensor
Dr Seth Hall

By
Eleanor Da Fonseca
School of Computing and Mathematical Sciences

Contents

List of Figures

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Acknowledgments

I would like to thank my primary supervisor, Dr Andrew Ensor, for his invaluable help and guidance. His knowledge and support during this thesis has helped me tremendously. I very much appreciate his patience in reading through my thesis several times and providing me with valuable feedback and corrections. A huge thank you for the countless hours spent helping me understand concepts, debugging code and testing as well as keeping me optimistic when things did not go as planned.

I would also like to thank my secondary supervisor, Dr Seth Hall. The work carried out in his PhD laid the foundation for this research. Without his knowledge and keen insight into the area of GPU-programming and computer vision this thesis could not have been written. I sincerely thank him for his friendship and willingness to help in any way he can.

Thank you very much to Boris Feron for taking the time to help proofread this thesis, for his moral support, advice and his supreme LaTeXskills. A big thank you to my parents for all the love, encouragement and assistance throughout my degree.

I would like to thank AUT for the several Summer Research Assistantship Awards that have helped fund my study and for the endless supply of coffee beans that allowed me to work long into the night. Lastly, I would like to thank everyone at the High Performance Computing Lab at AUT for providing such a fun and enjoyable work environment.

Copyright

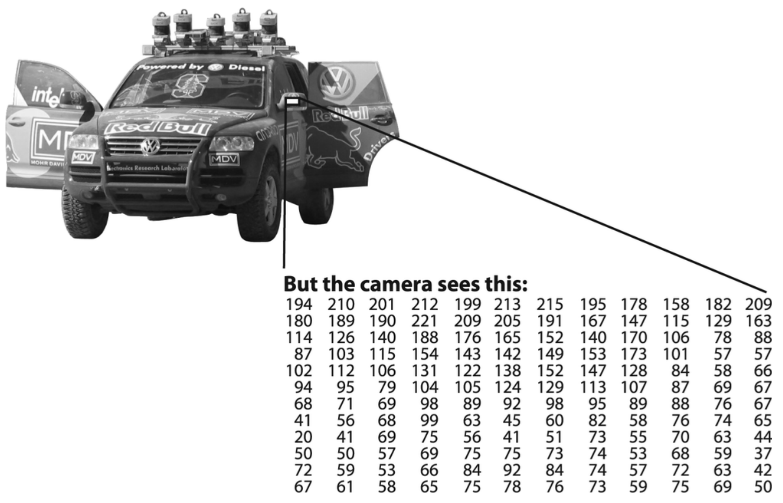
Abstract

ColourFAST is an alternative technique to FAST developed by Ensor and Hall used to extract feature point descriptors from an image based on colour change values. The extracted descriptor is compact and, therefore, efficient to compute and match. The purpose of this thesis is to extend the ColourFAST feature descriptor from a 4-dimensional vector to a 6-dimensional vector to improve feature point matching accuracy. This is achieved by incorporating spatial locality to gain a sense of the shape of an object alongside its colour change information. The main focus is designing, developing and testing feature point matching algorithms specifically architected for the GPU pipeline with an emphasis on accuracy while maintaining high throughput.

Chapter 1

Introduction

Vision refers to the ability to perceive and interpret the surrounding environment that is present in visible light through information processing by a visual system. Since humans are visual beings, the task of vision might seem trivial and deceptively easy. Humans are able to recognize objects and distinguish between over 30,000 categories of objects (like recognizing a person we know), locate objects in a space, track objects while in motion and coordinate our actions accordingly (for example, catching a ball during sports) and so on. Our brain responds to visual stimulus in a matter of milliseconds and is not hindered by changes in viewing conditions such as lighting and view point. Thus, our intuition would lead us to believe that vision is a simple task. However, there is a lot about visual systems that is yet to be understood. It, therefore, goes without saying that a machine attempting to "see and understand" visual data would be a much harder goal to accomplish (Sonka, Hlavac, & Boyle, 2007). Human vision system relies on our eye sensors receiving information, cognitive abilities processing and relaying feedback as well as several stages of processing to reach a decision that is made based upon years of experience. Computer vision does not possess these abilities; all it has to work off is a grid of numbers (see Figure 1.1), making it a fairly naive system that attempts to model the way human vision works although current systems are not nearly as sophisticated. Machine learning attempts to simulate this process but current systems have not yet reached


But the camera sees this:

```
194 210 201 212 199 213 215 195 178 158 182 209
180 189 190 221 209 205 191 167 147 115 129 163
114 126 140 188 176 165 152 140 170 106  78  88
 87 103 115 154 143 142 149 153 173 101  57  57
102 112 106 131 122 138 152 147 128  84  58  66
 94  95  79 104 105 124 129 113 107  87  69  67
 68  71  69  98  89  92  98  95  89  88  76  67
 41  56  68  99  63  45  60  82  58  76  74  65
 20  41  69  75  56  41  51  73  55  70  63  44
 50  50  57  69  75  75  73  74  53  68  59  37
 72  59  53  66  84  92  84  74  57  72  63  42
 67  61  58  65  75  78  76  73  59  75  69  50
```

Figure 1.1: What the computer sees
(Bradski & Kaehler, 2008)

the same level of sophistication. In addition, the data received by the system often contain noise and are affected by distortions due to imperfect lenses, mechanical imprecision, motion blur and issues in real world environments such as weather, lighting and reflections. These issues give us an insight into why computer vision is such a challenging field. There has been active research in this field for the past few decades; this overlaps with research in various other fields such as biological visual systems, machine learning, image processing, artificial intelligence and linear algebra. Computer vision lends itself to a wide range of potential applications and is, thus, a very interesting field that faces numerous obstacles. A few subfields include image processing, photogrammetry, 3D vision, optical flow and tracking. Examples of practical applications of computer vision systems include industrial robots, navigation via an autonomous vehicle, home surveillance systems, medical modelling and imagery, autonomous inspection and quality control used by manufacturing processes (Huang, 1996).

Computer vision is defined as "the transformation of data from a still or video camera into either a decision or a new representation" (Bradski & Kaehler, 2008). It deals with acquiring and processing images which are then analysed using statistical methods to extract data from them and obtain simple inferences based on individual pixel data. This data could be used in combination with other available information to help aid the system. Broadly speaking, computer vision can be described as the study of extraction of useful information from images by a vision system. The images could be a captured video sequence, views from multiple cameras or a still image. This area of study poses several challenges because vision is an inverse problem (Szeliski, 2010) where we attempt to retrieve meaningful information given insufficient, incomplete or noisy data to a particular problem. It often involves image analysis where the input is a 2D image that gets converted into a mathematical representation of some sort. Most computer vision tasks have very high computational demands and are very often required to work in real-time. This imposes further demands on vision algorithms as the data need to be processed on the fly.

Mobile devices are the modern day tangible embodiments of pervasive computing. The mobile phone market has seen tremendous growth since the 1990s with the trend continuing still (Want, 2010). In terms of research, the smart phone category of the mobile device market is very interesting as these devices are capable of providing functionality that desktop computers cannot. Most smart phones are equipped with high-quality graphics processing abilities, large memory, several high resolution cameras, high resolution displays, GPS systems and multiple sensors such as gyroscopes, accelerometers and proximity sensors. These features make mobile devices particularly interesting for a wide spectrum of computer vision applications. Computer vision can make intelligent use of the capabilities smart phones possess. A few examples of such applications include the work being carried out at Xerox PARC which aims to read vitals such as heart rate and respiration via a mobile device camera pointed at a face (*Computer vision and the future of mobile devices*, n.d.) and mobile-assisted driving that uses various sensors on the phone to aid the driver (Garcia-garrido et al., 2012).

As this is an area of active research, this thesis in particular deals with designing and implementing GPU-accelerated algorithms for real-time object recognition on mobile devices. It extends the feature descriptor, Colour-

FAST, proposed by Ensor and Hall from a 4-dimensional vector to a 6-dimensional vector by incorporating spatial locality information. It focuses on devising a feature point matching scheme that makes use of the shape of the object along with the ColourFAST colour change values. The algorithm is implemented and tested on a mobile device with a focus on achieving high accuracy in terms of matching while suffering a minimal performance hit. There are several popular smart phone platforms such as Android, iOS, BlackBerry and Windows Phone. For the purpose of testing the algorithms investigated in this thesis we chose the Android platform. However, the core algorithms are implemented using GLSL and are portable across mobile platforms that support OpenGL ES. Android is an open-source software stack developed by Google Inc. for smart phones and tablets. It includes a mobile operating system, middleware and applications. The core operating system is based on a modified version of the Linux kernel and is in written in C and C++. The user interface is based on direct manipulation via touch input and is, thus, primarily designed for touchscreen devices. Specialized interfaces have been also developed for other devices running Android such as Android TV, Android Auto and Android Wear. Applications are written in the Java language using the Android SDK and are executed on the *Dalvik Virtual Machine*.

The thesis is structured as follows. This chapter is intended to provide a general introduction to the field of computer vision and the use of smart phones for vision along with a brief summary of the work undertaken during this thesis. The second chapter provides background reading regarding mobile platforms, various vision tasks including popular techniques for feature description, detection and object recognition, graphics processing and the rendering pipeline. The literature review concludes with an in-depth discussion of ColourFAST feature points and its comparison to FAST feature points as the work in this thesis expands upon ColourFAST. The third chapter discusses the motivation behind extending upon the feature matching scheme used by ColourFAST by including spatial locality information and explains the work undertaken to achieve this. The fourth chapter elaborates on the GPU implementation of the new matching scheme clearly describing each render pass of the pipeline and its functionality. This chapter ends with an overview of the experiments conducted to test each phase of the algorithm. The next chapter reports and analyses the results obtained from several series of tests and investigates the effectiveness of expanding the ColourFAST feature point descriptor. The final chapter summarizes the work of this thesis and draws overall conclusions based on the results. The thesis concludes with appendices providing raw data used for modelling and analysis and the shader code implemented for the feature point matching scheme.

Chapter 2

Literature Review

This chapter covers related literature, background reading, used concepts and introduces the work undertaken. The topics discussed lay the foundation for this research and were the main areas of investigation during the thesis. It first covers mobile platforms and computer vision which is the main focus of this thesis. Computer vision applications and dominant techniques for feature detection, extraction and object recognition are discussed to give the reader a brief understanding of these study areas as these are later used for comparisons against our implementation. Next, GPU and GPU-based processing and computing are elaborated on as the algorithms implemented make use of the GPU to achieve high frame rates for computationally expensive tasks. This is a common theme across the entire thesis and each part of the algorithm is designed to specifically exploit the GPU pipeline architecture. The development of a GPU-based computer vision algorithm for object recognition is the backbone of our work. Contemporary computer vision applications based on recognition are discussed to show the relevance of the work conducted.

2.1 Mobile Platforms

During this thesis there were several competing smart phone platforms available for the development and deployment of application software such as the widely used Android and iOS platforms and the less popular Windows and Blackberry platforms. The Android platform is developed by the Open Handset Alliance led by Google and is based on the Linux kernel. Applications are primarily written using a subset of Java SE along with Android-specific API. Over the past few years the Android platform has become increasingly popular and currently holds the greatest market share in terms of mobile devices currently in circulation (*Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013*, n.d.). As Android is open source and not vendor specific it has become a popular choice for many mobile hardware companies such as Samsung, HTC, Motorola, LG, Huawei and others. Android applications are developed using the Android Development Tools (ADT) plugin for Eclipse or the more recent Android Studio which is a dedicated IDE for Android development and deployment.

2.2 Computer Vision

Computer Vision deals with the acquisition, processing and analysis of images to satisfy some goal. It often involves image restoration, object recognition, motion estimation and scene reconstruction. It transforms data received from a still or video camera into either a decision or new representation to accomplish some task such as detection, segmentation, localization etc. It can be considered a form of image analysis as it takes as input a 2D image and converts it into a mathematical description (Fung & Mann, 2004). It is considered to be the inverse problem of computer graphics as graphics produces image data from three-dimensional information whereas vision does the opposite. Thus, we see an overlap in technique between the two fields.

The *Open Computer Vision* library (OpenCV) is a cross-platform API originally developed by Intel and now an open-source project available for development of computer vision applications. It is the de facto library for developing real-time image processing and vision application software. It has an exhaustive collection of CPU-based implementations of popular vision algorithms for performing basic image and video I/O, image conversion, image processing, structural analysis, motion analysis, object tracking, pattern recognition, camera calibration, 3D reconstruction, view morphing, statistical classification, popular machine learning techniques and so on. The language used for development is either C/C++ or Python.

2.3 Image Segmentation and Feature Points

Computer vision consists of several tasks such as *Image Segmentation* which involves separating a digital image into multiple sets of pixels. This transforms the image into smaller sets of data that are easier to represent and analyse. It can be used during the first phase of locating objects or boundaries in an image. It can be thought of as describing each pixel using some descriptor such that pixels with similar descriptors tend to share similar appearance or characteristics.

Background Subtraction or foreground detection is a technique for isolating the object of interest in an image. Most often, the object being detected and identified such as cars, faces, signs etc. appear in the foreground of the image. During this stage of processing, parts of the image such as the background are eliminated so as to decrease the amount of processing done.

Feature Point Detection computes abstractions of image information and identifies unique areas or pixels in the image that exhibit characteristics such as pose invariance, distinctiveness, locality and repeatability. At each pixel, we either retain or discard the pixel depending on whether or not it shows those characteristics. This results in a subset of the image domain that can be further used for image processing. There is no universal definition of what a “good” feature is and is most often dependent on the problem domain and type of application. This is usually the first step following any image preprocessing for many computer vision algorithms (Wang, 2007). *Harris corner detection*, proposed in 1988, is the most widely used feature point detector. It is based on the calculation of eigenvalues of the second-moment matrix (Harris & Stephens, 1988). Even though very frequently used, it is not scale-invariant. Lindeberg first introduced an automatic scale-selection method in (Lindeberg, 1998) based on the determinant of the Hessian matrix as well as the Laplacian to detect blob-like parts of an image. The technique was refined by Mikolajczyk and Schmid in (Mikolajczyk & Schmid, 2001) to create a robust scale-invariant detector that was highly repeatable. This method used the determinant of the Hessian matrix to select the location of the point and the Laplacian to determine the scale. With a focus on speed, Lowe approximated the Laplacian of Gaussian (LoG) by a Difference of Gaussians (DoG) filter.

Feature Description or extraction is the task of representing the features chosen from the previous step. It forms the core of many computer vision algorithms such as object recognition, scene reconstruction and camera localization. A feature descriptor must be chosen based on the information needed for the task at hand. Some vision tasks require a high level of detail from the feature descriptor at the cost of increasing the volume of data to process. Other tasks might not be able to handle large amounts of data as performance might be important. In such cases, a compact descriptor might be chosen. The demand for feature descriptors to be fast to compute and match within the constraints of limited resources are increasing.

This thesis focuses on feature point detection and description and uses these features to identify objects in an image. As this is the main concern of the thesis, in the next section we shall discuss dominant feature point detection and extraction methods.

2.4 Feature Detection-Description Schemes

A large number of feature descriptors have been proposed including Gaussian derivates, complex features, moment invariants and steerable filters. However, descriptors that use smaller-scale features within interest point neighbourhoods as described in (Lowe, 1999) have shown to outperform the others as they encapsulate a significant amount of information about the

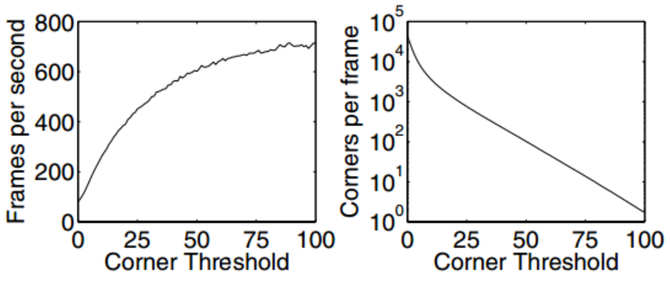


Figure 2.1: Speed of corner detection and number of corners vary with the corner threshold

(Rosten & Drummond, 2005)

spatial intensity patterns in an image providing a robust descriptor.

2.4.1 FAST

Corners are often used in vision systems as feature points as they are distinctive parts of an image. Many feature detection algorithms such as Moravec, Harris-Stephens, Wang-Brady and SUSAN all rely on corner detection. *Features from Accelerated Segment Test* (FAST) is a popular corner detection algorithm developed by Edward Rosten and Tom Drummond (Rosten & Drummond, 2005). The key advantage of using FAST over other techniques is its computational efficiency and speed that allows for on-line operation of a tracking system. It works by taking 16 pixels in a *Bresenham circle* of radius 3 around the centre pixel p where a corner is detected if at least N (typically taken to be 12) contiguous pixels have intensity differing from p above or below some threshold t as shown in Figure 2.1. Corners are then categorised as either *positive* or *negative* depending on whether the pixels are greater or smaller than p. Partitioning the corners in this manner is useful as positive feature points do not need to be compared to negative ones. Once corners have been detected, non-maximum suppression is used around each of the corners to eliminate adjacent neighbours that were picked as corners. The strongest (i.e. the one with the greatest intensity difference between it and its neighbours) is typically retained and the rest discarded.

2.4.2 BRIEF

Binary Robust Independent Elementary Features uses binary strings as an efficient feature point descriptor. It has been proven to be highly discriminative despite using relatively few bits and is computed from simple intensity difference tests. It employs the *Hamming distance* calculation which is very efficient to compute as opposed to the L_2 norm that is more often used. Thus, BRIEF descriptors are very fast to compute and perform matching with. It is worth noting that BRIEF descriptors do not provide rotational invariance and are thus not as useful as other methods such as SURF on datasets that contain rotations. However, in certain situations it does tolerate a small amount of rotation as seen in the test results in (Calonder, Lepetit, Strecha, & Fua, 2010).

2.4.3 SIFT

Scale-invariant Feature Transform is a feature detection and description algorithm published in 1999 by Daniel Lowe (Lowe, 1999). It is the most appealing descriptor for practical applications and is thus the most widely used. SIFT is a combination of three steps namely: key point localization, feature description and feature matching. First, SIFT applies Gaussian filters and then calculates the scale-space minima and maxima in the *difference of gaussian* (DoG) to locate the key points in an image. DoG is a greyscale image enhancement algorithm which involves subtracting a blurred version of the original image from another less blurred version. This operation can be computationally expensive hence key points are estimated separately from orientations and magnitudes of the points. Once these key points have been identified and stored in a database, they are individually compared to each key point identified in the new scene, and based on the *Euclidean distance calculation* of their feature vectors a match estimate is obtained. A subset of these key points that agree on the object detected in the new image along with its scale, orientation and location are used to pick good matches. Consistent clusters are found via an efficient implementation of the generalized Hough transform using hash tables. This subset is then further filtered for outliers and finally the probability of the presence of a particular object is computed. Objects that successfully pass all the above tests are correctly identified as a known object. It computes a histogram of oriented gradients around key points of interest and stores the bins in a 128-dimensional vector. To reduce dimensionality and increase speed of computation, PCA-SIFT has been proposed (Ke & Sukthankar, 2004) which produces a 36-dimensional descriptor. The increase in computational performance for matching is gained at the cost of discriminative power as shown by Mikolajczyk and Schmid. Another variant of SIFT, termed *GLOH*, has been experimented with (Mikolajczyk & Schmid, 2005) yielding even more distinctive feature points with the same number of dimensions but it proved to be more computationally expensive.

In conclusion, SIFT has the advantages of being distinctive, robust and relatively fast but its high dimensionality is a major drawback.

2.4.4 SURF

Speeded Up Robust Features coined SURF is a novel scale- and rotation-invariant feature point detector and descriptor. From (Bay, Tuytelaars, & Gool, 2006) we observe that with respect to repeatability, distinctiveness, and robustness SURF approximates or even outperforms previously proposed methods while being faster to compute and compare with. This is done using integral images for image convolutions, using the key strengths and insights gained from existing techniques and simplifying them. SURF aimed to be fast to compute while not sacrificing performance and accuracy, striking a balance between the descriptor's dimensionality and complexity versus being distinctive enough. The results show that on benchmark image sets and on a real object recognition application, the detector and descriptor are faster, more distinctive and equally repeatable (compared to (Lindeberg, 1998), (Lowe, 1999), (Ke & Sukthankar, 2004) and (Mikolajczyk & Schmid, 2002)). Another variant called *upright SURF* or U-SURF can be used if rotational invariance is not required resulting in a scale-invariant only version of the descriptor. This results in a performance boost as well as an increase in discriminative power between features.

The detection phase of SURF is based on the Hessian matrix (and the determinant of the Hessian matrix to identify location and scale) and uses a basic approximation (similar to SIFT's approach) while integral images are used to reduce computation time. SURF's descriptor uses a distribution of Haar-wavelet responses within the interest point neighbourhood of integral images for the sake of speed. Due to its size (64-dimensions) its computation time and matching is faster than SIFT while increasing robustness. The increase in overall robustness is achieved by using an indexing step based on the sign of the Laplacian, a novel technique described in (Bay et al., 2006).

The descriptor extraction phase works by determining a reproducible orientation from the information gathered from a circular region around the point of interest. A square region is then constructed in alignment to the orientation from step one from which the descriptor is extracted. The upright version skips the first step to find the orientation and is therefore faster to compute and well suited for applications where object rotation is not needed as the camera remains more or less horizontal.

Orientation Assignment

To achieve rotational invariance, each point identifies a reproducible orientation. This is done by calculating the Haar-wavelet responses in the x and y direction in a circular neighbourhood of radius $6s$ around the pixel under consideration (where s is the current scale at which the wavelet responses are computed). Therefore, it is obvious that the wavelets are big at higher scales and thus require using integral images for fast filtering. As only six operations are required to compute the wavelet at any scale, this computation is very quick. After this, the wavelet responses are weighted with a Gaussian centered at this point. The orientation is determined by calculating the sum of all the responses within a sliding orientation window covering an angle of $\frac{\pi}{3}$. The longest vector obtained from the sum of the horizontal and vertical responses within the window gives the orientation of the point.

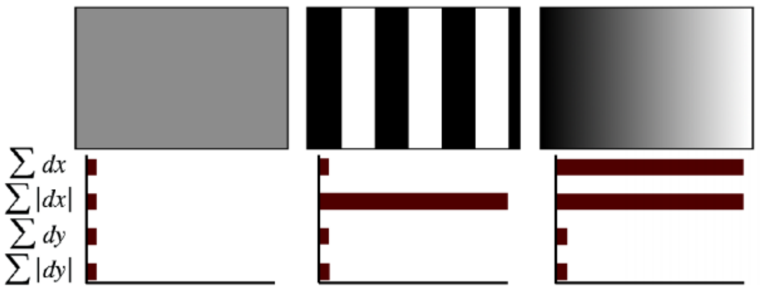


Figure 2.2: The descriptor entries of a sub-region represent the nature of the underlying intensity pattern. Left: In case of a homogeneous region, all values are relatively low. Middle: In presence of frequencies in x direction, the value of $|d_x|$ is high, but all others remain low. If the intensity is gradually increasing in the x direction, both values d_x and $|d_x|$ are high.
(Bay et al., 2006)

Feature Descriptor Components

In order to extract the feature point descriptor, the first step is to construct a square centered around the feature point oriented in the direction obtained from the previous step. U-SURF skips this step along with the orientation assignment step making it computationally much less expensive. The size of the window is chosen to be $20s$. The square region is split up into 4×4 square subregions and the Haar wavelet response in the horizontal (d_x) and vertical direction (d_y) are summed up over each subregion after weighting them with a Gaussian centered at the point of interest. This sum forms the first set of entries for the descriptor. To capture information about the polarity of change in intensity, the sum of the absolute values of the above responses is calculated. Each subregion now has a four-dimensional descriptor vector v $= (\Sigma d_x, \Sigma d_y, \Sigma|d_x|, \Sigma|d_x|)$. This gives a highly distinctive 64-dimensional vector for the entire square. The descriptor achieves invariance to contrast through normalization. Figure 2.2 shows how the descriptor behaves for three very different image intensity patterns within a subregion of an image. Such local intensity patterns combined with others would produce highly distinctive descriptors.

Descriptor	Recognition Rate
SURF-128	85.7%
U-SURF	83.8%
SURF	82.6%
GLOH	78.3%
SIFT	78.1%
PCA-SIFT	72.3%

Table 2.1: Feature point-based matching comparisons

SURF was tested based on feature point repeatability against four standard databases provided by Mikolajczyk (*Robotics Research Group*, n.d.) comparing results with dominant techniques including GLOH, SIFT and PCA-SIFT based on a similarity threshold and the nearest neighbour ratio. SURF outperformed the other descriptors in both cases in a systematic and significant way with a 10% improvement at times. Another test was performed aimed at recognizing objects of art in a museum under various conditions changes such as extreme lighting changes, objects behind reflecting glass, viewpoint changes, scale and rotation changes and different camera specifications. The matching was conducted using the nearest neighbour ratio matching strategy. An interest point in the target image is compared to an interest point in the database image by computing the Euclidean distance between the descriptor vectors of the two points. If the distance is found to be less than 0.7 times the distance of the second nearest match, it is said to be that object. A similar approach to matching is followed in this thesis. An alternative version of SURF termed SURF-128 was also tested which has double the number of descriptor values. It results in more discriminative power while maintaining comparable computation time. However, matching against the 128-dimensional vector is a lot slower. The results are given in Table 2.2.

SURF proves to be a fast and well performing feature point detection and description method that outperforms contemporary dominant techniques in terms of speed and accuracy. It can be easily extended to accommodate affine invariant regions and is therefore of particular interest in this thesis.

2.5 Object Recognition

Object recognition deals with some of the most important tasks of any vision system, namely detecting and recognizing objects. Humans can perform complex vision tasks in a fraction of a millisecond. The performance and accuracy achieved by computer vision systems still cannot compare to this. However, research done in this area in the past few years has seen tremendous progress. The obstacles faced by vision systems include heavy or partial occlusion and change of appearance. The main task of any recognition system is:
Given a database D of known objects and a test image I

1. Determine which (if any) of the objects in D appear in I
2. Determine the pose of the object

The choice of technique for an object recognition system depends on several factors namely:

1. How general is the problem?
   (a) Is it a 2D or 3D problem?
   (b) What is the range of viewing conditions?
   (c) Is there any contextual information about the data available?
2. What sort of data representation is best suited to the problem?
   (a) Local 2D features (SIFT, SURF)
   (b) 3D surfaces
   (c) Images (template matching)
3. How large is the search space (number of objects in the database)?
   (a) Small: possible brute force approach
   (b) Large: sophisticated search methods (search trees)

2.5.1 Image-based Recognition

Image-based recognition works based on the principal that if we *see* the object from every viewpoint and under all lighting conditions, then recognizing the object consist of merely a table lookup in the space of 2D images. If we consider an image I as a point in a space and all other such points generated by "viewing" I in every possible situation as described above, then an object is some surface in the space of all images. However, in practice the problem is that images contain a lot of information and there is an infinite variety of viewing conditions. In addition, objects in the image might be surrounded or occluded by other objects. Therefore, the data requires compression to filter out unnecessary information, the search space needs to be reduced and objects that are not of interest need to be segmented out of the image.

A simple approach to object recognition is *template matching* where the information assigned to certain pixels are used to compare the object to the pixels in the image of a template. This returns a value at each pixel depending on how close a match was obtained which is then used in various statistical techniques to perform the matching process. Some of the methods include:

Square difference matching where the sum of the squares of the differences between the template and the image intensities is calculated at each pixel (x, y) in the image, given by:

$$\sum_{m=0}^{w}\sum_{n=0}^{h}\big(f(x+m, y+n) - g(m,n)\big)^2 \tag{2.1}$$

where a perfect match would give zero and larger values indicate worse matches.

Correlation matching where the sum of the products of the template and the image intensities is calculated at each pixel (x, y) in the image, given by:

$$\sum_{m=0}^{w}\sum_{n=0}^{h} f(x+m, y+n) \cdot g(m,n) \tag{2.2}$$

so a perfect match would result in a large positive value and smaller values represent worse matches.

Correlation coefficient matching is similar to correlation matching except that the template intensity values and the image intensity values are taken relative to their mean value so a perfect match would result in the value 1 and smaller values indicate worse matches. These techniques normally involve normalization to eliminate the effect of lighting changes.

A more sophisticated approach to object recognition would be *back projection* which uses the distribution of pixel values in the template to perform matching as opposed to matching each pixel in the template with the corresponding pixels in the image as is done in template matching. This technique uses the rectangular patches in the template against the target image by calculating how well the pixel value fits with the distribution of values in the histogram created from the colour channels of the image. The method is particularly convenient for matching textures such as grass or human flesh which do not necessarily have a uniform colour but do have a reasonably consistent distribution of colours.

Many sophisticated object recognition techniques come under the field of machine learning where the algorithm progressively refines the parameters it utilizes in a *classifier* function as it evaluates training data. The algorithm adapts its behaviour as it learns from data, resulting in increasingly accurate results over time. The training data typically is a large collection of images that have been labeled and had important features extracted such as edges, contours or pixel colour distribution. The classifier then uses these extracted features from the image to estimate some attribute for the image. For example, a particular classifier could be used to determine the presence of a face in the image. Machine learning makes use of various statistical techniques depending on time and memory available to train the classifier and how quickly the classifier is expected to process new images. Some of these include the *naive Bayes classifier*, *decision trees*, *boosting*, *neural networks* and the *Haar classifier*.

2.6 Machine Learning

Bradski and Kaehler defines the goal of *machine learning* (ML) as turning data into information. After learning from a collection of data, we would like the machine to answer questions about the data. For example, what other data resembles this the most? Or is there a face in this image? Machine learning is a scientific discipline that investigates the development and study of algorithms that are able to learn from previous data by extracting rules or patterns from that data and turn it into information. A learning algorithm would typically work on data such as temperature values, DNA sequences, colour intensities etc. *Features* are then extracted from the data set and used to construct a model to learn from. To achieve the particular goals for a task, machine learning algorithms analyse the extracted feature values and adjust weights, thresholds and parameters to obtain the best results possible for our original goal. The term *learning* refers to this process of parameter and threshold adjustment to fulfill our goal.

2.6.1 Types of Learning

Learning is divided into two broad categories - *supervised* and *unsupervised* learning (see Figure 2.3). If the feature vector data has a label associated with it, it is referred to as supervised learning as the label may be used to "teach" the algorithm about the data set. However, if we wish to observe how the data forms groups on its own without any associated label, unsupervised learning might be utilized. Supervised learning can be divided into *classification* which deals with categorical labels sets such as learning to associate a name to a face and *regression* where the data labels are numeric or ordered. Regression tries to fit a numeric output based on some numeric or categorical input. In comparison, *clustering algorithms* are used when the data available is not labelled and are interested in seeing what groups

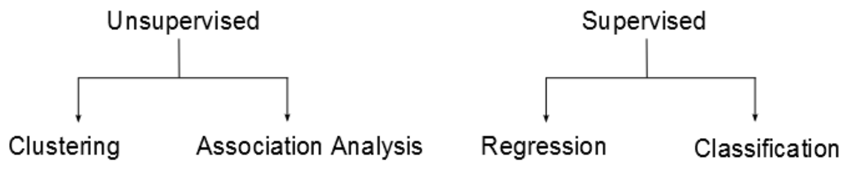


Figure 2.3: Types of Machine Learning

the data naturally falls into. The aim is to group unlabelled feature vectors that are determined to be close to each other by some chosen measure of closeness.

Typically, developing a classification system follows the steps out lined below:

1. Split the original data into a large training set and smaller validation and test sets. The test set is not used during training. Thus, the tests are conducted on data that has not been "seen" by the classifier before.

2. Run the chosen classifier over the training set to learn the model given the extracted data feature vectors.

3. While training the classifier, smaller tests are conducted against the validation sets. This helps tweak weights and thresholds until the performance is acceptable.

4. Next, the classifier is tested against the test set.

5. Its results against the test set are recorded and if it does poorly, more feature data might be added or another classifier might be chosen.

The choice of classifier is largely dependent on a number of factors as there is no universal "best" classifier. A classifier might perform well or poorly based on considerations such as computational constraints, data and memory available and time need to train the classifier. As such, we shall discuss one such popular classifier that is widely used for a number of applications.

2.6.2 Boosting

Boosting falls under the category of discriminative classifiers and was introduced by Freund and Schapire in 1997. The overall classification is done by combining weighted classification decisions from a group of weak classifiers. These weak classifiers tend to be very simple on their own. They usually consist of *single-variable* decision trees called *stumps* or at most up to a few levels of splits. They are each trained individually during the training phase where the stump learns its decision from the feature data. Based on the accuracy of their performance after the training, they each get assigned a weighted vote for their contribution to the final decision-making. The data used is a collection of labelled input feature vectors associated with a scalar label. The algorithm starts out with a data point weighting which is used to penalize the algorithm for misclassifying a point. The most characteristic features of boosting is that as the algorithm advances, the penalty value evolves so as to allow the following weak classifiers to focus on the points that were misclassified while previous classifiers were being trained. This continues until the total error count for the group of classifiers is below a certain threshold. This is a particularly effective method when the amount of training data available is large and the system has sufficient time to train.

2.7 Graphics Processing Unit

The graphics processing unit, or *GPU*, is a specialized processor designed to offload 3D graphics rendering from the central processing unit. Modern GPUs can carry out computer graphics processing very efficiently since they are architected to accelerate single-precision floating-point arithmetic operations such as matrix multiplications on geometric data like vertex attributes that are processed independently of each other and in parallel, thus achieving a very high number of *floating point operations per second* (flops). Most devices such as embedded systems, personal computers, gaming consoles and mobile phones have an integrated GPU. Thus, rather than relying on the more general purpose central processing unit, certain tasks are offloaded to the GPU. It is commonly used for creating lighting effects, texturing objects, generating dynamic shadows and reflection effects, performing culling operations and producing animation effects each time a 3D scene is redrawn. These tasks are highly computationally intensive and would put tremendous strain on the CPU. The first GPU was developed by Nvidia Inc. in 1999 who marketed the *GeForce 256* GPU which is a "single-chip processor with integrated transform, lighting, triangle setup/clipping and rendering engines that are capable of processing a minimum of 10 millions polygons per second" (*GPU: Changes Everything*, n.d.).

2.8 Overview of OpenGL

Accessing the graphics hardware on devices requires an interface such as OpenGL or Direct3D which provide an API for rendering graphics.

OpenGL is a cross-language, multi-platform software interface that can be used to access the graphics hardware on a variety of devices. It was first introduced in 1992 and has become the industry's most widely used and supported API for 2D and 3D vector graphics. It is supported on a wide range of operating systems including Mac OS, Windows, Linux, Unix and so on. It is a hardware-independent interface governed by the OpenGL Architecture Review Board (ARB), based on the C programming language. OpenGL ES is a subset of OpenGL for use on embedded systems. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies. It is typically used to gain access to the GPU to achieve *hardware-accelerated graphics rendering.* The process of transforming geometric objects provided by a software application into a two-dimensional object to be drawn on the screen is called *rendering.* There are two common software API for graphics rendering namely OpenGL and Direct3D which is part of the proprietary DirectX API developed by Microsoft for the Windows platform for 2D and 3D graphics rendering.

Android supports graphics development via OpenGL ES and is thus used in this thesis for implementing and testing various computer vision algorithms.

2.9 Rendering Pipeline

The *rendering pipeline* is the series of steps that OpenGL takes when rendering geometric objects to obtain a 2D raster representation of a 3D world on the screen (*Rendering Pipeline Overview*, n.d.). It accepts vertex attributes (vertex coordinates, normal vectors, RGBA colour, texture coordinates) as input as well as read-only state configuration values (such as enabled or disabled states, model-view and projection matrix) and produces as output pixel-based data (such as pixel colour, depth or stencil values). The implementation and optimization of the graphics processing pipeline on a GPU might vary between vendors but the effects of the pipeline are always equivalent to the seven stages listed below. The application first sets up the input to send to the pipeline (using OpenGL commands) which is an ordered list of vertices and vertex attributes. This input is then processed as follows:

Per-vertex operations accepts as input the attributes for an individual vertex. It uses the model-view matrix, projection matrix and texture matrix to transform the vertex attributes into *clip space coordinates* and assigns each vertex a primary and secondary colour. This stage performs light shading calculations if enabled.

Primitive Assembly assembles the vertex data for each vertex until it has sufficient vertices for a complete primitive. Points, lines, triangles, quads or polygons can be rendered and the number of vertices required varies for each primitive type.

Primitive Processing clips the primitive from the previous stage against the view frustum either rejecting the primitive if it lies completely outside the view frustum or allowing it to progress through the pipeline if it lies completely inside, or else clipping away parts that are not visible. Once the appropriate clipping and culling is performed, it converts the clip space coordinates into *window coordinates.*

Rasterize takes the window coordinates and colour for each vertex in the primitive and rasterizes the primitive. This results in a pixel representation of the primitive as *fragments* which is a pixel-sized square inside the viewport region with an associated colour, depth and other attributes such as texture coordinates.

Fragment Processing performs texture mapping using the fragment colour determined by the previous stage and the active texture for each texture unit. Other effects such as fog, secondary colour etc. are also used to modify the colour at that stage.

Per-fragment Operations performs a series of simple tests to determine whether the fragment should result in a pixel at window coordinates or instead be rejected. It uses the *pixel ownership test* to check whether or not the fragment would result in a pixel occluded by an overlapping window, the *alpha test* to check whether the fragment colour's alpha value satisfies the alpha comparison and other similar tests to eventually output the pixel's x and y coordinates along with a colour value. After these tests have been completed, the colour of the fragments that were not culled are blended with the colour at the corresponding location in the frame buffer.

Frame Buffer Operations Last of all, the fragment data are written to the framebuffer. These pixels might remain in the pipeline until the GPU has collected a sufficient number of processed primitives sending a group of pixels to the buffer.

The above stages describe the fixed functional rendering pipeline which is very efficient for traditional graphics rendering, allowing a variety of advanced rendering effects to be achieved by the use of pipeline configurations. However, controlling these configurations add to the complexity of the pipeline and hinder throughput. Moreover, certain effects are not possible using the fixed function pipeline, e.g. refractions and soft shadows. This has led the modern GPU pipeline away from a fully fixed function approach to a pipeline which allows application generated code to replace some parts of the pipeline rather than merely configure it. Thus, the application can take control of how vertices, primitives and fragments are processed and manipulated by the GPU.

2.9.1 Programmable Shaders

A *shader* is a piece of code that gets deployed to a specific stage of the pipeline and replaces that stage. There are several shader languages that have become dominant for programmable parts of the pipeline like Renderman, OpenGL Shading Language, High-Level Shader Language and Cg. Most GPUs allow three types of programmable shaders to be used which get deployed to the specific part of the pipeline they are to replace.

Vertex Shader replaces the fixed functionality of the Per-vertex Operations stage of the pipeline, operating on each individual vertex received by the pipeline and transforming it

Geometry Shader replaces the Primitive Processing stage of the pipeline operating on each primitive, possibly changing its type, introducing or removing vertices and outputting one or more primitives to progress through the pipeline

Fragment Shader replaces the Fragment Processing stage of the pipeline operating on each individual fragment it receives, possibly changing how a texture is mapped onto the fragment or some other modification to the colour of the fragment.

For this thesis, several vertex and fragment shaders were developed to emulate the fixed functionality of the pipeline processing performing various image processing tasks rather than its original functionality described above.

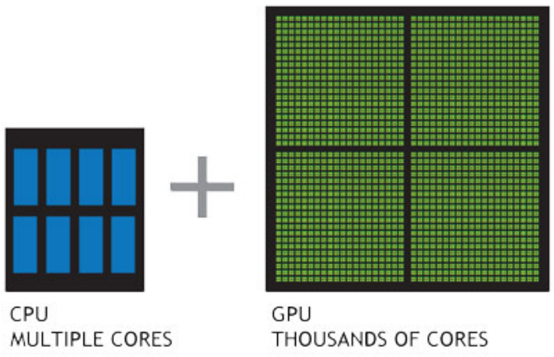


Figure 2.4: CPU versus GPU cores
(*What is GPU Computing?*, n.d.)

2.10 GPU Accelerated Computing

Computer vision algorithms can be very computationally intensive as each pixel might need to be processed to extract information. These algorithms also need to be able to run in real-time (Fung & Mann, 2005). This is quite a strain for the CPU as it is not optimized for such algorithms. However, the architecture of the GPU allows for this strain to be shared by its many cores.

The trend has been for GPUs to progressively open up more and more of the graphics rendering pipeline to allow programmable shaders to execute at stages within the pipeline. As a result the potential for GPU computing beyond graphics rendering has increased tremendously. It has found a place in diverse fields such as machine learning (Raina, Madhavan, & Ng, 2009), scientific and medical image processing, linear algebra (Kruger & Westermann, 2003) and statistics (Liepe et al., 2010). This is termed as *general-purpose computing on graphics processing units* (GPGPU) or *GPU-accelerated computing*. It refers to using the GPU for performing computations besides rendering which were traditionally handled by the CPU. "GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications" (*What is GPU Computing?*, n.d.) (see Figure 2.5). As

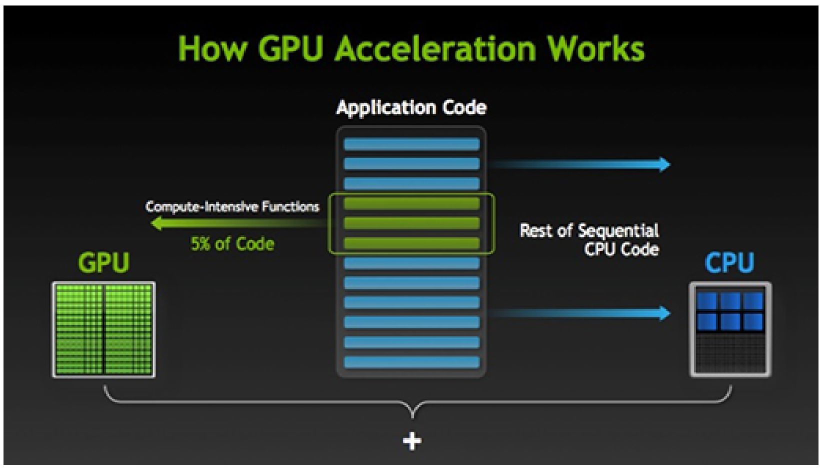

How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
GPU
Rest of Sequential CPU Code
CPU
+

Figure 2.5: How GPU Acceleration Works
(*What is GPU Computing?*, n.d.)

GPU are architectured specifically for performing single-precision floating-point arithmetic operations (such as matrix multipications) on streaming data (such as vertex attributes) that are processed independently in parallel they have been able to achieve a very high number of *floating point operations per second* (flops). For example, the ATI Radeon or an Nvidia Tesla GPU can achieve up to approximately 4 teraflops (single-precision floating point), whereas a six core Intel CPU achieves approximately 80 gigaflops. Thus, the GPU has become increasingly attractive for computations that are arithmetically intense where the data can be processed independently. The processing nature of the GPU lends itself very well to certian types of algorithms that can exploit the streaming data parallelism design. However, multicore CPUs have a much larger memory cache on the processor for faster memory access and offer better handling on unpredictable branches and looping. So the CPU stills tends to outperform the GPU with algorithms that require task parallelism where multiple tasks execute in parallel with little inter-process communication.

2.11 ColourFAST

This section discusses in detail ColourFAST feature points which are the basis of this thesis. This feature point descriptor was proposed in (Ensor & Hall, 2013) and is a real-time GPU-based feature detection and descriptor algorithm. It extracts vector-based feature strength and direction measures from the RGB colour channels of an image. As shown in the previous sections, certain computer vision tasks benefit greatly from GPU acceleration; the algorithm has been architected with this in mind. This method has shown great results for the purpose of object tracking and very basic object recognition with an implementation on mobile devices. It has shown several improvements over FAST which has also been tested on mobile devices. From the analysis in (Ensor & Hall, 2011) we see clearly see that mobile GPU-based algorithms tend to be advantageous to image processing in terms of speed which is of utmost importance for applications that rely on high framerate output. GPUs have become popular for many image processing tasks due to their superior performance with highly parallel floating-point calculations and the nature of such tasks as described previously (Kim, Park, Cui, Kim, & Gruver, 2009).

Mobile devices face numerous challenges when it comes to computer vision applications such as varying quality and resolution in camera capture, differing processing capabilities leading to low frame rates, non-standard formats for image and video capture on different devices and so on. Most feature detection and description algorithms have been designed for CPU-based applications and their performance on mobile devices is unacceptable for an application that requires real-time performance. This led to compromises such as offloading most of the image processing to a networked server, introducing markers into the scene (such as fiduciary markers or QR codes) or using predetermined templates. The aim of ColourFAST was to develop a robust feature detection and description algorithm that could be used for real-time image processing tasks such as tracking and recognition without the need for the above mentioned workarounds. It proposed a variant of FAST specifically designed for the GPU that is quick to compute and can be described compactly. As the dimension of the feature point descriptor has a direct impact on the time taken to compute and search for the descriptor, it was of utmost important to ensure that ColourFAST was compact. It attempted to resolve certain issues that FAST faced such as feature points not being detected consistently between frames, some corners not being detected due to thresholding of greyscale images and issues due to the presence of noise. To eliminate these issues, colour channels were used to provide valuable information about the interest points. Due to the *single instruction multiple data* (SIMD) nature of GPUs, this added information did not contribute to increased computation time. In addition, the change in colour across the pixel gave an orientation for the feature so a direction measure was added. The combination of Bresenham colour values and direction gave a unique and compact 4-dimensional feature vector. To counteract the presence of noise the implementation performed a 3×3 smoothing step.

ColourFAST follows the same approach as FAST which is an efficient and simple corner detection algorithm for greyscale images. FAST takes 16 pixels in a Bresenham circle of radius 3 around the pixel being tested where at least 12 of these pixels should have an intensity differing from the centre pixel above some threshold for that pixel to pass the test and be considered a corner. Due to the nature of computing on GPUs, the thresholds used by FAST were no longer needed. Therefore, the minimum requirement of 12 pixels in the neighbourhood of the pixel being tested was removed to allow any feature point to be considered (not just corners). The intensity difference threshold was also removed which proved to be beneficial.

2.11.1 Half Bresenham and Feature Strengths

This technique follows a similar method to FAST where it calculates feature point values at a pixel by subtracting it from the average of the neighbouring pixels in a Bresenham circle to give the change in intensity at that pixel. The difference between the two is that ColourFAST computes the intensity change in each channel rather than on a greyscale image. It also eliminates the thresholding which is common in FAST to get rid of points that are not

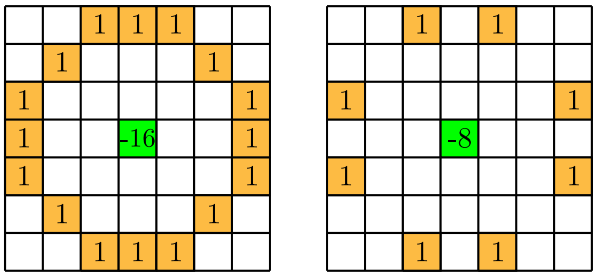


Figure 2.6: Bresenham circle (left) used by FAST and half-Bresenham circle (right) used by ColourFAST
(Ensor & Hall, 2013)

corners. In order to boost performance, it reduces the number of texture lookups performed by the GPU to half by using 8 neighbouring pixels rather than 16 (as in FAST). Before the feature strength calculation is performed, a smoothing step is used to blend each pixel with its 8 neighbouring pixels. This allows only half the adjacent pixels to be used for the calculation and makes this approach more robust to noise.

The calculation at each pixel is given by Equation 2.3

$$[H]\begin{pmatrix}F_R\\F_G\\F_B\end{pmatrix}=\begin{pmatrix}P_R\\P_G\\P_B\end{pmatrix}-\sum_{i=0}^{n=8}\begin{pmatrix}N_{i,R}\\N_{i,G}\\N_{i,B}\end{pmatrix}\times\frac{1}{8} \tag{2.3}$$

After computing the feature point strength values for a pixel, each colour channel contribution to the final result can be adjusted to give an overall strength value for the pixel as a scalar. The amount each channel contributes to the overall strength value can be controlled via a *weighting factor* which is chosen through experimentation. The value picked for the weighting depends upon the channel that is of particular interest. The strength value for a pixel is particularly useful for eliminating weaker pixels that are not of interest. However, for extraction of the feature descriptor, a three dimensional vector was found to provide richer information about the point of interest.

		9	12	18	12	9		
		12	16	24	16	12		
9	12	18	12	18	12	18	12	9
12	16	12	-72	-96	-72	12	16	12
18	24	18	-96	-128	-96	18	24	18
12	16	12	-72	-96	-72	12	16	12
9	12	18	12	18	12	18	12	9
		12	16	24	16	12		
		9	12	18	12	9		

Figure 2.7: Neighbourhood pixel contributions to ColourFAST
(Ensor & Hall, 2013)

2.11.2 Feature Orientation

This phase of the calculation computes a θ value for the pixel under consideration. This value gives the direction of change of intensity for the given pixel. It is given as the arc tangent of the x direction and y direction. The orientation of a point is calculated by taking the vector sum of the eight RGB colour changes from the previous step and subtracting values from the pixels that lie below the centre pixel from the ones that lie above to give the ΔY. Subtracting the left pixel values from the right pixel values gives ΔX. Using the Pythagorean theorem and assuming a distance of 1 unit from the centre, we obtained two constants in the X and Y directions to multiply each RGB value in the corresponding directions. We then combine the vector obtained for each colour component to give a single overall value by taking the dot product with a unit vector formed from the feature point strength value. This gives a heavier weighted orientation for colour components with more drastic changes in intensity (see Figure 2.8). Both the above phases combined gives us a four dimensional vector.

$$X_{dir} = \begin{pmatrix} F_R \\ F_G \\ F_B \end{pmatrix} \cdot \begin{pmatrix} \Delta X_R \\ \Delta X_G \\ \Delta X_B \end{pmatrix} \Bigg| \begin{pmatrix} F_R \\ F_G \\ F_B \end{pmatrix} \Bigg| \tag{2.4}$$

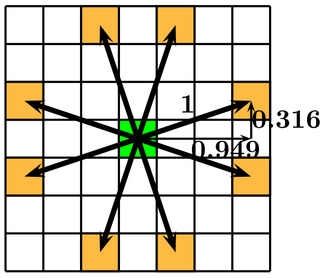


Figure 2.8: Feature orientation vector calculations (Ensor & Hall, 2013)

$$Y_{dir} = \begin{pmatrix} F_R \\ F_G \\ F_B \end{pmatrix} \cdot \begin{pmatrix} \Delta Y_R \\ \Delta Y_G \\ \Delta Y_B \end{pmatrix} \Bigg| \begin{pmatrix} F_R \\ F_G \\ F_B \end{pmatrix} \Bigg| \tag{2.5}$$

$$\theta = \arctan \frac{Y_{dir}}{X_{dir}} \tag{2.6}$$

ColourFAST has been shown to have several advantages over many popular feature detection and description algorithms as it uses the three colour channels rather than a greyscale image. This exposes features that show a change in colour but not necessarily in overall intensity. The extracted description is a mere four dimensions compared to either 64 or 128 dimensions as used by most techniques and is fast to compute and match with. Thresholding is not performed as edges that are not corners can be strong feature points and provide valuable information for object tracking or recognition. All the calculations are performed with the GPU in mind and optimized for vector SIMD calculations. Thus, little or no perfomance penalty is incurred as seen in Chapter 5 of (Hall, 2014).

Testing was performed on the Samsung Galaxy S2 I9100 (ARM Mali-400 MP4 GPU) and the Samsung Galaxy S4 GT-I9505 (Adreno 320 GPU). These devices run Android v2.3 and 4.2 respectively and use the Open GL ES 2.0 pipeline with GLSL version 1.0. Through these tests, it was proven that high frame rates can be obtained running the ColourFAST detection and

Device and Resolution	FAST (OpenCV)	FAST (GPU)	ColourFAST (GPU)
Galaxy S2 (640x480)	25.1	30.5	39.8
Galaxy S2 (800x480)	20.6	25.5	32.4
Galaxy S4 (640x480)	21.3	53.7	51.4
Galaxy S4 (1920x1080)	8.3	23.3	21.3

Table 2.2: Feature point throughput comparisons (FPS)
(Hall, 2014)

description calculations every frame (for results, see Table 2.2). ColourFAST was shown to extract richer information from features compared to FAST while maintaining comparable performance.

2.12 Example Application Domains

2.12.1 Medical Image Analysis

The progress made in computer vision techniques over the past decades has allowed extensively improved diagnosis, treatment and predication of diseases via medical imaging (C. Chen, 2014). With the aid of texture, contour, shape and contextual information from sequences of images computer vision can provide rich 3D and 4D data to help medical professionals. Vision techniques such as image segmentation, machine learning, pattern recognition and scene reconstruction provide powerful tools that greatly benefit trained medical specialists. With the growing amount of annotated medical data, large-scale, data-driven methods are apt to bridge the semantic gap between images and medical diagnosis. The emphasis of this field is on collating, organising and learning from large-scale medical imaging data sets. Techniques that work well on previously unseen images and that can be applied and scaled to large data sets are of particular interest. Such methods must be robust to weak or noisy annotations in training data. For example, these can be used for (*MICCAI Workshop on Medical Computer Vision: Algorithms for Big Data (bigMCV)*, n.d.):

1. Anatomical structure localization through object recognition and cat-

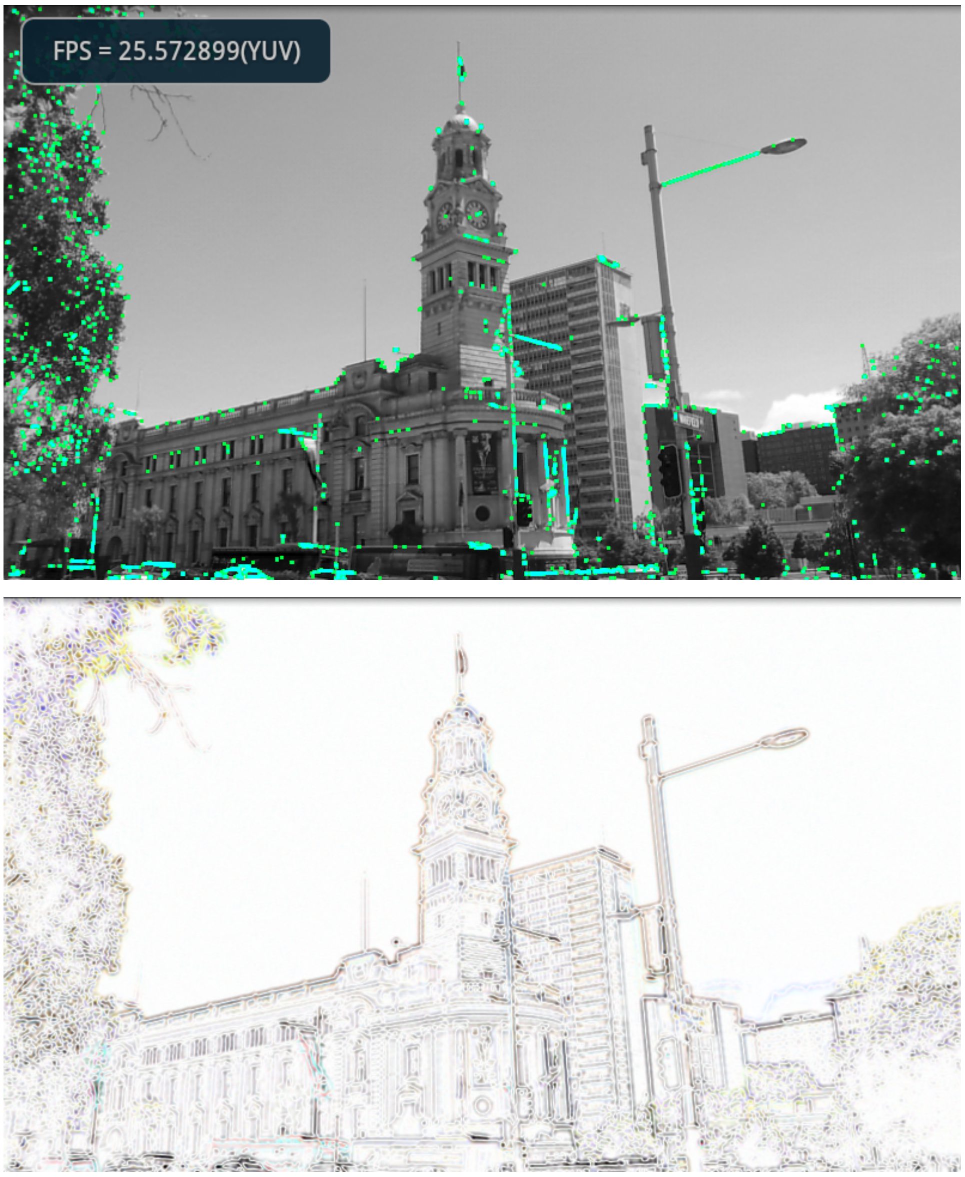


Figure 2.9: Outdoor scene with FAST (upper) versus ColourFAST (lower) feature values evaluated at each pixel
(Ensor & Hall, 2013)

egorization

2. Developing 3D image descriptors and interest points for object localization
3. Generative models of 3D image scenes relying on, or complementing, population atlases of anatomy or function
4. Features and algorithms dealing with image acquisition variations, such as CT scan plan or MR pulse sequence variations, with/without contrast agents

Tommasi, Torre, and Caputo describe the success rates achieved with extensive experiments using *support vector machines* and *spin glass-Markov random fields* (both machine learning techniques) for skin lesion classification. Malignant melanoma is a fast spreading, complex disease that is incurable in its advanced stages. Therefore, early detection and treatment are the key factors in reducing occurrences. Detection of skin lesions via Epiluminescence Microscopy is the most widely-used diagnostic tool. However it is prone to misinterpretation and misidentification by dermatologists. As a result, an autonomous system for melanoma detection and recognition would be very valuable support for physicians. There have also been numerous studies using segmentation and feature extraction and performing classification with a nearest neighbour classifier such as (Ganster et al., 2001). A mean recognition rate of 61% was achieved and these results constitute the state of the art in the field (Tommasi et al., 2006). Grana, Pellacani, Cucchiara, and Seidenari have also conducted work in this area and proposed mathematical descriptors for the border of pigmented skin lesion images and evaluated their efficacy for distinguishing them from other lesion groups. Data mining techniques in conjunction with vision based methods were explored in (Grzymala-Busse, Grzymala-Busse, & Hippe, 2001) with great success. Lefevre, Colot, Vannoorenberghe, and de Brucq proposed a theory based on data fusion, regression and classification (see Section 2.6.1) called the *Dempster-Shafer's theory.* They applied their classification process on a training set of 81 lesions: 61 benign and 20 malignant melanoma

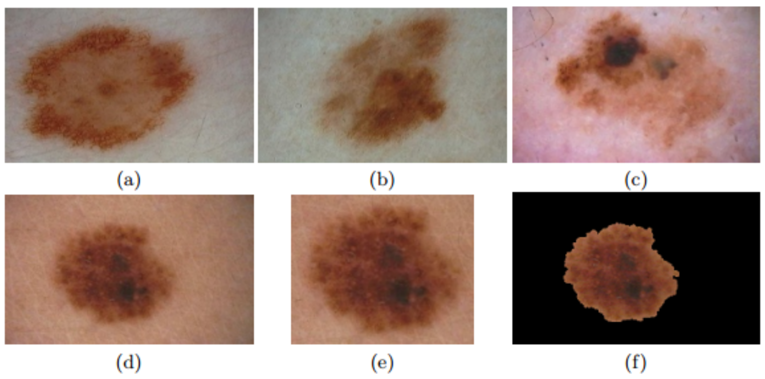


Figure 2.10: Examples of skin lesions images used: (a) image of a benign lesion, (b) image of a dysplastic lesion,(c) image of a malignant lesion. (d) shows an example of an entire image, (e) the same image hand-segmented, (f) the same image mask-segmented
(Tommasi et al., 2006)

lesions.

2.12.2 Augmented Reality

The term *augmented reality* (AR) refers to augmenting the view of a real-world environment by computer generated input such as image, sound, video or GPS data. It combines the user's environment (obtained via a camera feed) with computer generated models to produce a powerful user interface technology. Using AR technology along with computer vision techniques such as object recognition, the surrounding environment of the user becomes interactive and digitally manipulable. Compared to *virtual reality* which replaces the user's real-world, AR blends the physical and virtual world by registering 3D or 2D graphical information to real-world locations rendered to a display in real-time (Lee, Kitayama, Kwon, & Sumiya, 2009), (Reitmayr & Schmalstieg, 2003). This has given rise to a number of new applications such as AR games, task support, medical imagery and surgical guidance, education, navigation and travel assistance and so on. With this, the tools and SDKs for developing such applications have also grown. Following are some of the popular tools for augmented reality applications.

Layar Browser is a free mobile application available for a variety of mobile platforms developed by the Dutch company Layar. It allows user-selected location-based *layers* to be displayed within the Layar browser using *points of interest*. These points of interest could be images, 2D or 3D objects, animations or an associated action such as a URL, phone number or email address.

Junaio is a free MAR browser and SDK for the development of third-party MAR applications for Android and iOS developed by the German company Metaio. It allows developers to create *location-based channels* to have text, animations, or static 3D objects as points of interest. The developer also has the option of using optical *Glue channels* for previously registered images to be recognized and overlayed with animated or static models. As with Layar, a point of interest can also hold audio, video, images or URL links.

2.12.3 Autonomous Vehicles

An autonomous vehicle (or driverless car) is an automated vehicle that fulfils the main transport capabilities of a traditional car (Krogh & Thorpe, 1986). It can sense its environment and navigate through it without any human input with the help of radar, lidar (a remote sensing technology using a laser), GPS and computer vision (Gehrig & Stein, 1999). So far, driverless cars exist as prototypes and research demonstrations with the first autonomous cars were introduced by Carnegie Mellon University's Navlab and ALV projects in 1984 and Mercedes-Benz and Bundeswehr University Munich's EUREKA Prometheus Project in 1987. Since then, several other companies and research groups have undertaken work in this area. Prominent among them are Mercedes-Benz, General Motors, Bosch, Nissan, Toyota, Audi, Oxford University, Google and Vislab from University of Parma. In July 2013, Vislab introduced BRAiVE, a driverless vehicle that autonomously navigated through a mixed traffic route that was open to public traffic in a highway and urban setting (refer to Figure 2.11). The greatest challenge faced was negotiating roundabouts of varied sizes and shapes, underpasses, pedestrian crossings and traffic lights (*Public ROad Urban Driverless-Car Test 2013*, n.d.). It made use of:

1. Two frontal cameras used to locate obstacles, obey traffic lights, identify road markings and reconstruct the terrain
2. Lateral cameras combined with lateral laser scanners to handle merging traffic and manoeuvring roundabouts
3. Frontal laser scanner along with the two lateral scanners to locate lateral objects
4. Two back-facing cameras that were used to identify vehicles in adjacent lanes

The vision system used by BRAiVE is based on the real-time processing of two images obtained from two synchronized cameras and provides terrain estimation in front of the vehicle while also locating and tracking frontal obstacles. The image processing by the system is done at the rate of 12.5Hz from each camera which are each 1 megapixel. This allows for scene reconstruction of a 3D world in front of the vehicle every 80 milliseconds.

As of 2013, Florida, Nevada, California and Michigan have passed laws allowing autonomous cars on the road. Several European cities such as Belgium, France, Italy and the UK plan to run transport systems for autonomous cars. Spain, Germany and the Netherlands currently allow driverless cars to be tested in traffic. The *DARPA Grand Challenge* and the *DARPA Urban Challenge* are competitions for autonomous vehicles organized and funded by the Defense Advanced Research Projects Agency. The vehicles are expected to obey the state driving laws, be entirely autonomous using only information obtained via sensors and GPS. They must also be able to operate in varying weather conditions such as rain and fog and must avoid obstacles along the way. The challenge has attracted the attention of numerous universities, businesses and research organisations (*DARPA Urban Challenge*, n.d.).

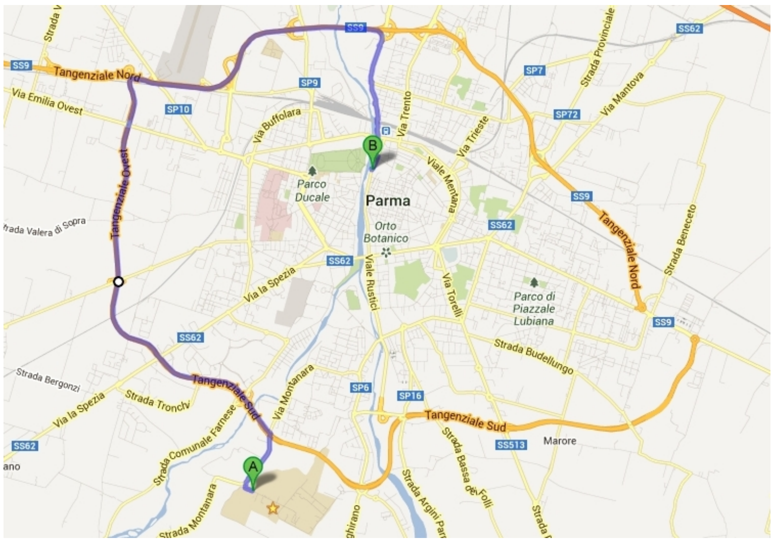


Figure 2.11: Course navigated during the PROUD-Car Test (*Public ROad Urban Driverless-Car Test 2013*, n.d.)

2.12.4 Object Recognition

Google Goggles

Google Goggles is an object recognition mobile application developed by Google that allows the user to do a Google search by taking a picture with their mobile phone (*Search for pictures with Google Goggles*, n.d.). Google Goggles is specifically developed to run on the Android operating system. This application works best on object categories such as books and DVDs, famous landmarks, logos, contact information, artwork, businesses and products and text. However, the algorithms used are not publicly known. It can be used for the following:

- Scan barcodes using Goggles to get product information
- Scan QR codes using Goggles to extract information
- Recognize famous landmarks
- Translate by taking a picture of foreign language text

- Add Contacts by scanning business cards or QR codes
- Scan text using Optical Character Recognition (OCR)
- Recognize paintings, books, DVDs, CDs, and just about any 2D image
- Solve Sudoku puzzles
- Find similar products

Chapter 3

Design

3.1 Motivation

The work undertaken in this thesis follows on from (Hall, 2014) and similarly focuses on the efficient computation and matching of feature points for the purpose of object recognition on mobile devices. ColourFAST feature points have proven to be sufficiently unique for the purposes of tracking as the feature vectors do not vary much between consecutive frames. For the purpose of simplistic object recognition against a small database they appeared to be adequately distinctive yielding promising results. However, the matching accuracy drops when tested on more complex real-world objects. This is due to the fact that real-world objects tend to vary a lot in terms of scale, rotation, skew, lighting conditions and so on. As the original points rely solely on colour change, the descriptor contains no information about the scale or rotation of the object. In addition, it cannot differentiate between objects of similar colour variations. For example, it would struggle to tell the difference between the two squares shown in Figure 3.1 as, of the four feature points in each, two are identical and the other two only differ in their direction component by a quarter turn. This work attempts to enhance the descriptor by adding spatial locality information alongside the feature strength and direction measures. The work undertaken benefits greatly from the work carried out previously as the feature points are pro-

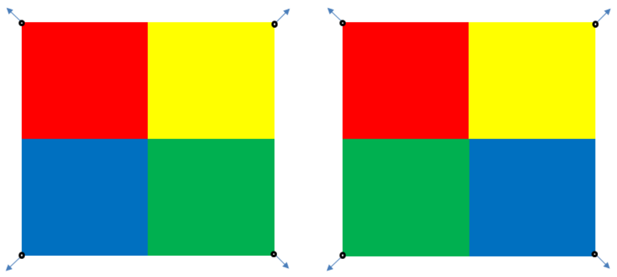


Figure 3.1: Two objects with same RGB colour changes but different direction measures in two feature points

cessed through the GPU pipeline very quickly and, thus, lends itself well to real-time feature point extraction, description and matching.

Due to the use of direction measures calculated for each feature point, it might be able to tell the squares apart in some situations. However, with structures that have similar colours and similar direction measures it is completely arbitrary which one gets matched best (for example, see Figure 3.2). In this figure each feature point has the same colour change values as well as the same orientation and therefore the two objects cannot be differentiated. By augmenting the feature descriptor with spatial locality information associated with each feature point, we are able to gain an understanding of the shape of the object. This gives us an idea of the relative position of each feature point in the object. This work focuses on using this information in conjunction with the RGBθ values to develop a scale and rotation estimator that can be used to correct the position of the feature points and improve upon the feature matching results.

3.2 Spatial Locality

As seen above, using colour change values in each of the RGB channels along with an orientation measure may not have enough discriminative power when it comes to feature matching between real-world categories of objects that have tend to have similar colour changes. Thus, the feature vector is extended by adding the x and y coordinates for each feature point chosen.

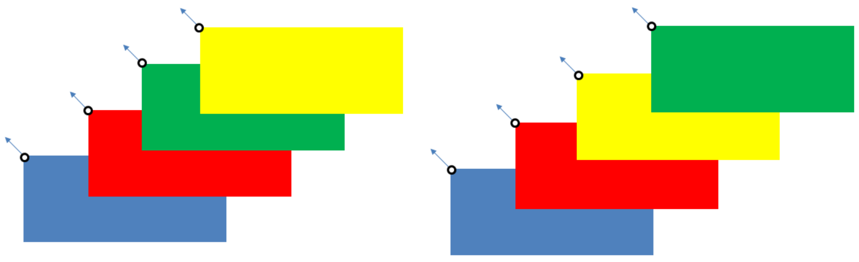


Figure 3.2: Two objects with same RGB colour changes and same direction measures

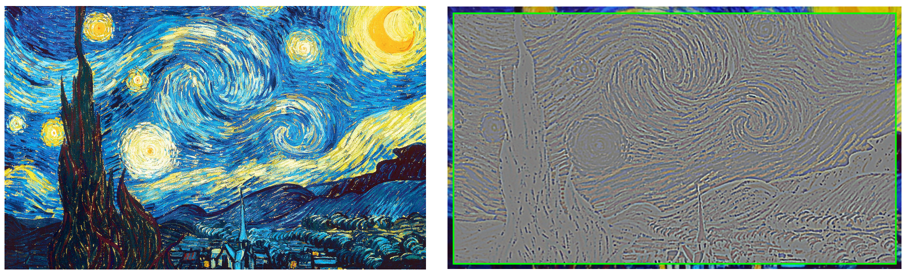


Figure 3.3: The Starry Night, Vincent van Gogh (1889)

This gives a 6-dimensional vector which is very compact, quick to calculate and perform matching with. It is important to note that the feature descriptor does not rely on dimensionality reduction techniques such as *Principal Component Analysis*, *Local Discriminant Embedding* (H.-T. Chen, Chang, & Liu, 2005) or quantization to encode the floating-points into integers using fewer bits as most other description schemes do. This makes it particularly efficient to compute and match. Since the colour change is being calculated for a given ColourFAST feature point, it does not incur a performance penalty to also extract the coordinates for that point.

Another drawback that was noticed in (Hall, 2014) was the low repeatability of feature points. Since the feature points were selected manually (by touching the screen at the desired location) if the user selected a different feature point while testing, the match accuracy tended to drop. Moreover, the original feature vector would not be able to tell that the objects in Fig-

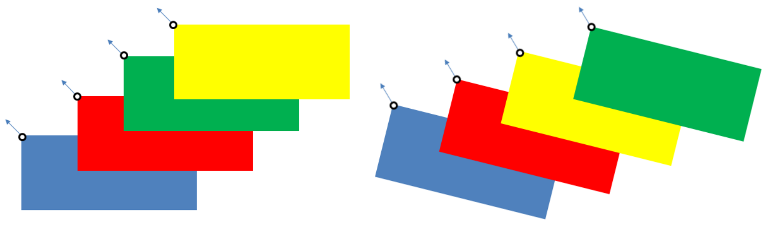


Figure 3.4: Two objects with the same RGB colour changes and same direction measures with one slightly rotated

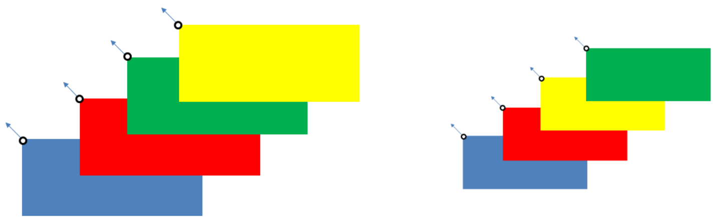


Figure 3.5: Two objects with the same RGB colour changes and same direction measures with one scaled

ure 3.4 and 3.5 were in fact the same object but at a different rotation and scale respectively. By introducing spatial locality to the descriptor, we aim is to address some of these issues.

3.3 Scale and Rotation Estimator

Using the example above, it is clear that based on colour change there would be no sense of rotation or scale and each of the objects in Figure 3.2 might be considered the same object. Developing a scale and rotation estimator provides richer information regarding the objects in a scene. The estimator starts by picking the two best matching feature points (referred to as *anchor points*) based on the Euclidean norm of their $RBG\theta$ values between the target object and the reference object. The θ component is weighted to contribute less than the RGB components so that the matches are based mainly on colour change and are, therefore, less sensitive to rotations. Once the anchor points have been selected, they are used to estimate the pose of the object during the feature matching phase.

Let $T = \langle r_T, g_T, b_T, \theta_T, x_T, y_T \rangle$ be a feature vector from the target image and $D = \langle r_D, g_D, b_D, \theta_D, x_D, y_D \rangle$ be a feature vector from the database being matched. We get the distance between the orientation components of two points by using the formula:

$$d_\theta = 1 - 2 \times \big||\theta_T - \theta_D| - 0.5\big| \tag{3.1}$$

where θ_T and θ_D are encoded between 0 and 1 (rather than in degrees or radians) to conveniently allow their values to be passed between the CPU and GPU and between multiple render passes.

The Euclidean distance between the RGB components of two points is given by:

$$d_{RGB}(T, D) = \sqrt{(r_T - r_D)^2 + (g_T - g_D)^2 + (b_T - b_D)^2} \tag{3.2}$$

We define a modified and weighted version of the Euclidean distance:

$$d_{RGB\theta} = d_{RGB}(T, D) + d_\theta \times 0.15 \tag{3.3}$$

which gives an overall distance measure for two points. The closest matches based on this distance d are used as anchor points as they are assumed to be the same point on the target as in the database image (possibly scaled and rotated). To pick two point as anchors, we match every feature point vector on the target image to every feature point vector in the database of objects and compare the match values obtained.

Once the two anchor points $(x1_T, y1_T)$, $(x2_T, y2_T)$ in the target image and $(x1_D, y1_D)$, $(x2_D, y2_D)$ in the database image have been picked, we assume that $(x1_T, y1_T)$ is the same point as $(x1_D, y1_D)$ and $(x2_T, y2_T)$ is the same point as $(x1_D, y1_D)$, $(x2_D, y2_D)$. The ratio of their Euclidean distances gives the scale factor s for the target image.

$$s = \frac{d\big((x1_D, y1_D), (x2_D, y2_D)\big)}{d\big((x1_T, y1_T), (x2_T, y2_T)\big)} = \frac{d_D}{d_T} \tag{3.4}$$

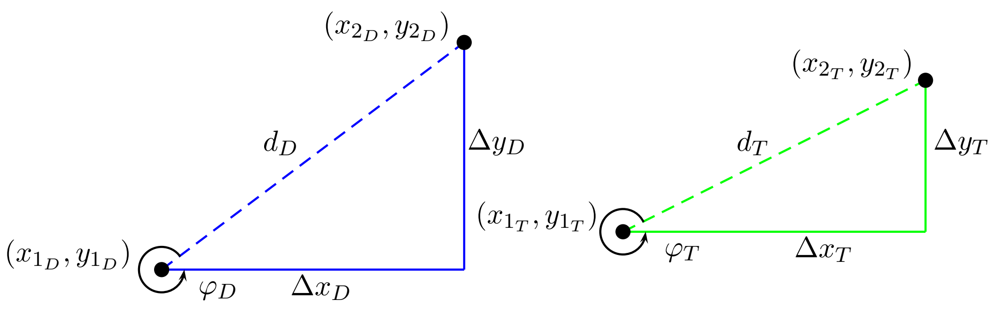


Figure 3.6: Scale and rotation calculations

For the rotation estimate, we calculate φ_D, φ_T as follows:

$$\varphi_D = \arctan \frac{\Delta y_D}{\Delta x_D} \tag{3.5}$$

$$\varphi_T = \arctan \frac{\Delta y_T}{\Delta x_T} \tag{3.6}$$

$$\varphi = \varphi_D - \varphi_T \tag{3.7}$$

The values s and φ are then used to scale, rotate and translate the object in the target object to match the object in the database. In linear algebra, linear transformations can be represented by matrices. If T is a linear transformation mapping $\mathbb{R}^n$ to $\mathbb{R}^m$ and $\vec{x}$ is a column vector with n entries, then

$$T(\vec{x}) = A \times \vec{x} \tag{3.8}$$

for some $m \times n$ matrix A, called the transformation matrix of T.

For transforming the target object in two dimensions, the following transformation matrix is used

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = s \cdot \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x - x_{1_T} \\ y - y_{2_T} \end{pmatrix} + \begin{pmatrix} x_{1_D} \\ y_{1_D} \end{pmatrix} \tag{3.9}$$

The translation is performed with respect to the first anchor point $(x1_D, y1_D)$ in the database. This point corresponds to the best possible match between the feature points in the target image and those in each object in the database and is, thus, used for the translation. As we iterate through each feature point in the target image, its x, y values are "corrected" using the above transformation matrix to obtain a predicted location (x', y') for that point. As explained previously, this approach builds on the assumption that the two anchor points in the target image and database respectively are indeed the same point. To make this approach more robust, using an additional number of anchor points might be considered.

Chapter 4

Implementation

4.1 GPU-based ColourFAST

This chapter discusses the implementation of the feature point description scheme proposed in (Hall, 2014) including the extension by adding spatial information. It also explains the method used for feature point matching and describes the tests conducted. All of the algorithms implemented were set up and run on devices running the Android platform. However, minor changes in the set up would allow the program to run on any device that supports OpenGL ES.

ColourFAST starts by processing a coloured image frame and converting it from the Android default format (NV21) to the RGB colour space. The NV21 format produces YUV values for the image frame where the Y and the UV values are interleaved as shown in Figure 4.1. Therefore, the first task for the program is to perform a GPU render pass to output either YUV or RGB values in a single texture. The input for this render pass is a texture with Y values and another for the UV values and the output is a single texture that contains values in either of the chosen colour spaces. This is done so that the remaining render passes require only a single lookup texture for pixel colour values. The RGB colour space was chosen for this implementation as this is the same colour space used by Hall for the entire ColourFAST pipeline. However, in practice there was no significant advantage using either colour

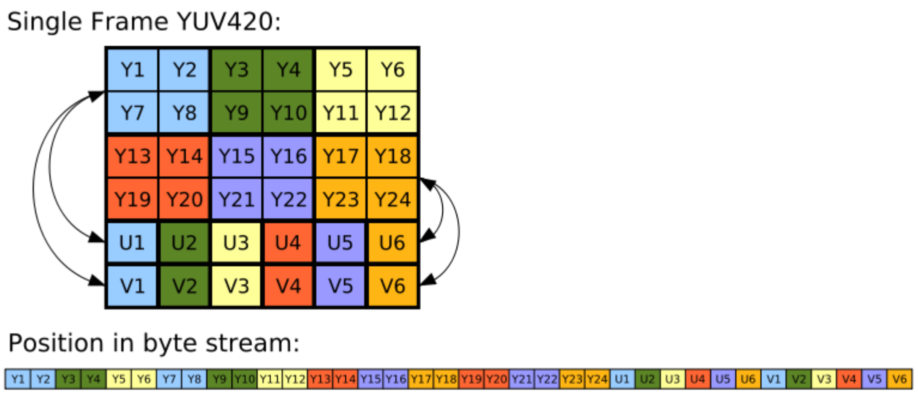


Figure 4.1: YUV colour space values in the NV21 format (default for Android devices)
(*YUV pixel formats*, n.d.)

space over the other.

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} Y + V \times 1.402 - 0.701 \\ Y - U \times 0.344 - V \times 0.714 + 0.529 \\ Y + U \times 1.722 - 0.886 \end{pmatrix} \tag{4.1}$$

The next two render passes perform a smoothing step using a 3×3 convolution kernel on each of the three colour channels. As the Gaussian kernel is separable it is applied as two separate one-dimensional convolutions as shown in Equation 4.2 in the X and Y direction. This requires an additional render pass but reduces the number of texture lookups as well as the number of multiplications that need to be performed, thus providing a small performance boost overall.

$$\begin{pmatrix} 0.09 & 0.12 & 0.09 \\ 0.12 & 0.16 & 0.12 \\ 0.09 & 0.12 & 0.09 \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.4 \\ 0.3 \end{pmatrix} \cdot \begin{pmatrix} 0.3 & 0.4 & 0.3 \end{pmatrix} \tag{4.2}$$

The following render pass uses the same approach as FAST to calculate the feature point values at each pixel. The value at each pixel is taken and subtracted from the average of the neighbouring pixels surrounding it within the Bresenham circle. This gives a measure for the change in intensity. The difference from the FAST calculation is that ColourFAST is performed in each colour channel as opposed to a greyscale image. The threshold used for FAST to determine a corner is eliminated allowing all points to be considered as feature points. The number of neighbouring pixels used in the calculation is half of that used by FAST (i.e. 8 surrounding pixels). Due to the smoothing step, using 8 surrounding pixels results in effectively using 65 surrounding pixels as their values are blended with the values of the pixels used. This results in a feature point value that is more robust to noise than a FAST feature point. The formula used for this calculation is given by Equation 2.3 on page 30. Thus, so far, we have obtained a three-dimensional feature vector.

The next render pass uses the output texture from the smoothing step to calculate the orientation for a given feature vector. This is obtained by taking the vector sum of the RGB changes for the 8 surrounding pixels, subtracting pixels below the centre pixel from the pixels above the centre to give ΔY and likewise subtracting horizontally to obtain ΔX. Using the $\arctan \frac{\Delta Y}{\Delta X}$ formula we obtain a θ value for the feature point in each channel, which are then combined together using a weighted average. This produces a four-dimensional feature vector giving the colour change in each channel along with the feature orientation. This vector will be referred to as RGBθ for the sake of convenience for the rest of this thesis. As textures passed into the GPU pipeline can have at most four components at each texel, this compact vector is particularly convenient since every feature point descriptor can be held in one single texel in the texture that gets bound for each render pass.

4.2 Feature Discovery via Contour Tracking

ColourFAST uses an algorithm based on contour tracking for discovering new feature points on an object. It starts by using a feature point that gets placed manually on some part of the contour of an object. From that point on, the algorithm progressively follows the contour and discovers new

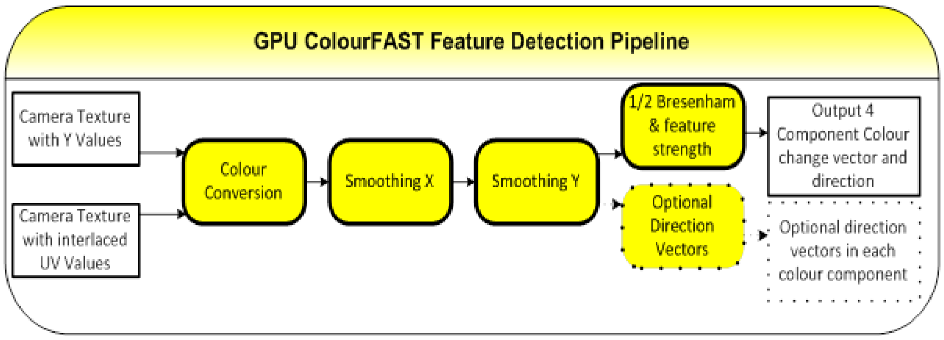

GPU ColourFAST Feature Detection Pipeline
Camera Texture with Y Values
Camera Texture with interlaced UV Values
Colour Conversion
Smoothing X
Smoothing Y
1/2 Bresenham & feature strength
Optional Direction Vectors
Output 4 Component Colour change vector and direction
Optional direction vectors in each colour component

Figure 4.2: GPU ColourFAST feature detection pipeline. Shaders are shown in yellow and the input/output textures are in white. The shader shown with the dotted border is optional for cases when a direction vector in all 3 components is desired

(Hall, 2014)

points that are distinctive based on their feature point strength measure. It utilizes Haar-like features (Viola & Jones, 2001) to track the ridges and valleys formed around the outline of an object due to the colour change in intensity. It continues to trace around the contour of the object until the desired number of features have been extracted. This method was used to discover feature points during the feature matching tests performed by Hall. These tests were carried out on a database of objects consisting of 50 popular company logos similar to that shown in Figure 4.3.

It is implemented using a combination of the CPU and GPU. The CPU keeps track of where on the contour of the object the algorithm has traced up to and the GPU executes a shader to discover new feature points. The input to this shader is the output texture from the ColourFAST feature detection phase of the pipeline containing the feature point vectors. Using the inner and outer ridges formed around an object, a Haar-like detector with the mask $(1, 2, 1, -1, -2, -1)$ is used to stay on the contour of the object. This mask is applied five times during this render pass across ten pixels while moving up to two pixels on either side of the ridges and valleys. The maximum absolute value obtained is used to progress the discovery

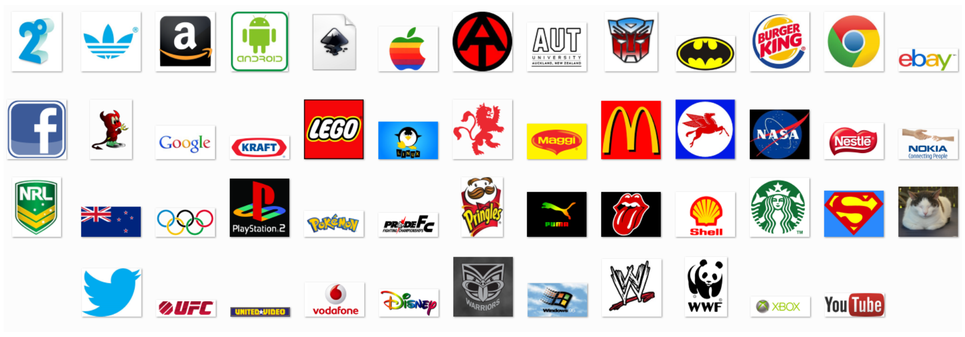


Figure 4.3: Database of objects used for feature matching

point along the contour. Once the strongest feature point has been found as described, the orientation can be determined which gives the direction in which to move the mask along as the algorithm progresses.

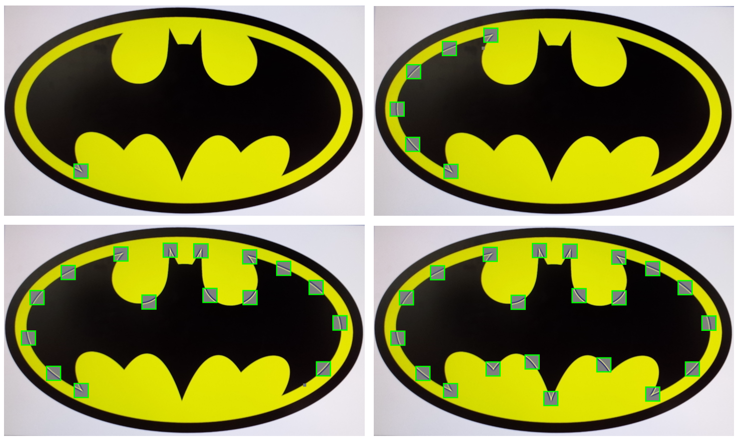
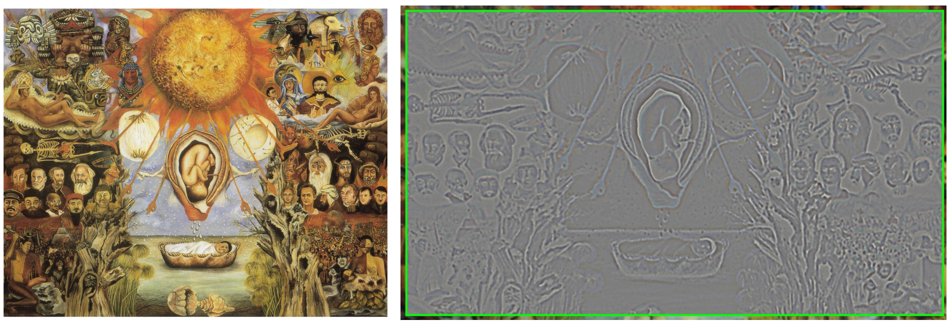


Figure 4.4: Feature Point Discovery via Contour Tracking

This method works well for tracking purposes as evidenced by the clustering and tracking results obtained. However, for feature matching and recognizing objects this technique is rather slow and often chooses feature points that are not on the contour of the object due to sudden movement of the camera, blurry edges, low contrast between the object and the background resulting in poorly defined contours etc. To speed up the feature extraction process and improve the contour tracking the CPU-side code and the shader were modified. The number of iterations the shader performed each frame was increased. Hence, rather than finding the strongest feature and moving along once each frame, we find the strongest feature among several iterations of the code from each frame. This produced a significant boost in performance and greatly benefited the testing carried out on the database of logos as the feature points used for matching no longer needed to be manually selected.

4.3 Automated Feature Discovery

The first phase of testing involved feature point extraction from each logo using the contour tracking algorithm from Section 4.2 and optimizing it specifically for logos using the method described. Feature points are extracted by first clicking on one edge of the logo. From that point, the algorithm follows the edge along the logo and finds feature points that are the local maximum so as to find distinctive points on edges that are not within some minimum distance from each other. This gives a wide spread of distinctive points on each logo. However, the contour tracking for feature discovery did not seem to work for real-world objects as the edges of the object were not as clearly defined as those on computer-generated images such as logos. In addition, real-world objects often contain intricate patterns that the tracking algorithm struggled with. As a result, a fully automated version of feature discovery was implemented which did not rely on contours. It starts by drawing a grid around the object of interest and finding all the points within that grid that are above a certain strength threshold. We begin with a grid of size 1100×620 pixels which amounts to $682,000$ feature points. However, since we are only concerned with strong feature points a threshold is used to eliminate feature points that do not meet our strength criteria. This thresholding is performed by the CPU whereas the feature point values are calculated on the GPU. After the threshold has been applied, we obtain approximately between $10,000$ to $200,000$ feature points depending on the image. These are sorted based on feature strength and among the highest features we use a distance threshold to only pick the



Original Image Image with feature points

Figure 4.5: Moses, Frida Kahlo (1945)

features that are some distance away from each other so as to spread out the selected feature points (see Figure 4.5).

We start by calculating ColourFAST values ($RGB\theta$) for the entire grid (the portion of the image shown in grey in Figure 4.5). Let P be a given point within the grid on the target image. The feature points are sorted based on feature point strength which is calculated as the follows:

$$FP_{strength} = \sqrt{|127 - P_R|^2 + |127 - P_G|^2 + |127 - P_B|^2} \tag{4.3}$$

The first step is to decode the P_R, P_G, P_B values (which are the RGB values of a given point) to between -127 and 127 after which we obtain the feature point strength.

We eliminate all feature points with strength less than a certain threshold t_s (initially set to 40 chosen through experimentation) which gives us a subset of feature points S. The following algorithm is run against each of these points and a greedy approach is used:

1. Put the strongest feature point into the result set T and remove it from S

2. Take the next feature point in S and compare it to all the points in T. If it has at least x distance from each of them, add it to T; else discard it.

3. Repeat the previous step until there are k number of feature points in T.
4. If all the feature points in S have been exhausted and T does not contain k feature points, we lower the value of t_s and x and repeat the steps.

This approach ensures that strong, distinctive feature points are selected over weaker ones. It also ensures that feature points are selected in different regions of the image rather than being clustered in one region of the image that might happen to have a high degree of colour change. The initial feature point strength thresholding is performed to reduce the amount of data to be sorted and tested; this produces a tremendous speedup. The entire automated feature discovery process could be offloaded to the GPU via an additional render pass to achieve a performance gain.

4.4 Spatial Locality

A scale and rotation estimator is introduced to the matching of ColourFAST feature points, making the process more robust and consistent. In order to achieve this some significant changes are made to the set up of the shaders. The first and second render pass match each point on the target object to every point in each database object to find the best two matches for that given target object point. These four points are used as our anchors (two points on the target and two from the database) to find the difference in rotation and scale between the points on the target object that in the database. Once the rotation and scale have been calculated we adjust the points on the target to the required rotation and scale and perform a match again with these transformed spatial values. This theoretically allows the points on the target to be at a different angle and scale than the points in the database and still match with the same accuracy.

The feature matching uses three multi-render passes as described below. Let f be a fragment within the geometry being passed in to the GPU and f_s and f_t be the s and t coordinates for the given fragment. Let $T =$

$\langle t_0, t_1, \cdots, t_{o-1} \rangle$ be the target image and $D_k = \langle d_0, d_1, \cdots, d_{n-1} \rangle$ be an object in the database D containing objects $\langle D_0, D_1, \cdots, D_{m-1} \rangle$.

4.4.1 Shader 1: Preliminary Matching

This render pass takes two textures as input:

1. the $RGB\theta$ feature vector in T (see Table 4.1)
2. the $RGB\theta$ feature vector for each object in D (see Table 4.2)

	t_0	t_1	$\cdots$	t_{o-1}
T	0.8000,0.8156,0.8000,0.078	0.2668,0.2313,0.2235,0.8705	$\cdots$	0.2941,0.2823,0.2392,0.4315

Table 4.1: $RGB\theta$ texture for the target image T where each entry is one feature point in T

	d_0	d_1	$\cdots$	d_{n-1}
D_0	0.9568,0.9764,0.9764,0.1882	0.9411,0.9960,0.9607,0.3764	$\cdots$	0.8470,0.8705,0.8705,0.0627
D_1	0.8588,0.8509,0.8352,0.1372	0.8627,0.8549,0.8039,0.1411	$\cdots$	0.8078,0.8235,0.7843,0.6823
D_2	0.8274,0.8588,0.8352,0.9333	0.8078,0.8078,0.7686,0.0352	$\cdots$	0.7176,0.7647,0.7568,0.0666
D_3	0.7921,0.7882,0.7960,0.0901	0.7882,0.7921,0.7843,0.1137	$\cdots$	0.7176,0.7058,0.6901,0.0431
D_4	0.8235,0.8431,0.8235,0.6470	0.7607,0.7803,0.7450,0.1450	$\cdots$	0.3529,0.3254,0.3411,0.0941
D_5	0.9254,0.9411,0.9411,0.1686	0.8470,0.8705,0.8627,0.8117	$\cdots$	0.7372,0.7333,0.6745,0.9803
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
D_{m-1}	0.8352,0.8392,0.8431,0.3490	0.8313,0.8352,0.8196,0.8666	$\cdots$	0.7568,0.7647,0.7411,0.9411

Table 4.2: $RGB\theta$ texture for the database where each row has the feature points for one database object

For executing the shader code the geometry passed in is a quad with height equal to the number of objects in the database and width equal to the number of feature points on the screen (as shown in Figure 4.6).

It is important to note that each texture coordinate in Figure 4.6 performs the exact same procedure described above and each of these run in parallel on a separate shader instance within the GPU. Thus, there is no intercommunication between each of these cells and all the textures bound as input are shared between all the shader instances.

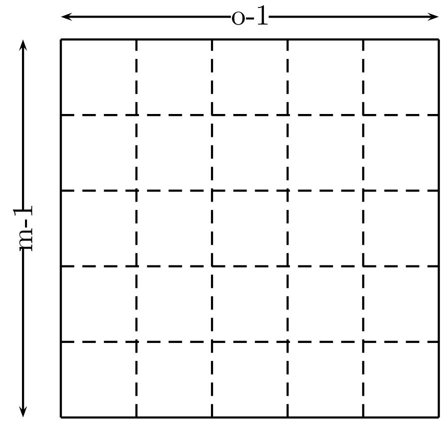


Figure 4.6: Geometry for 1st render pass

4.4.2 Shader 2: Rotation and Scale

The second shader takes the output from the previous render pass (Figure 4.7) as an input texture along with two other textures:

1. the x, y feature vector in T (see Table 4.3)

2. the x, y feature vector for each object in D (see Table 4.4)

	t_0	t_1	t_2	$\cdots$	t_n
T	0.659,0.370	0.496,0.320	0.621,0.429	$\cdots$	0.608,0.259

Table 4.3: x, y texture for the target image T where each entry corresponds to one feature point in T

As each texel can hold only up to four components, the $RGB\theta$ and x, y values must be placed in seperate textures. However, since the XY texture only contains two components the remaining two components could also be utilized without any modification to the way the shaders are currently set up if the descriptor needed to be further extended.

The geometry passed in is a quad with height equal to the number of objects in the database and width equal to 1 (as shown in Figure 4.8). This render pass executes an instance for each row of the texture output from the

```
Min-Distance($i, j, n, T_{RGB\theta}, D_{RGB\theta}$)
                                        ▷ Shader 1: Preliminary matching
                     ▷ $i, j$ are the row and column of the current fragment
$b \leftarrow 0$                                 ▷ distance between two points
$b_{min} \leftarrow 10$                                        ▷ lowest distance
$b_D \leftarrow 0$           ▷ database point at which lowest distance was found
$b_T \leftarrow 0$             ▷ target point at which lowest distance was found
for $k \leftarrow 0$ to $n-1$ do
                         ▷ where $n$ is the number of feature points in $D_k$
    $db \leftarrow$ Look up $d_{k_{RGB\theta}}$ at $\Big(k \cdot \big(1/(n-1)\big), j\Big)$
    $target \leftarrow$ Look up $t_{i_{RGB\theta}}$ at $(i, h/2)$
                              ▷ where $h = 1$ is the height of the $T_{RGB\theta}$
    $b = d(db, target)$
    if $b < b_{min}$ then
        $b_{min} = b$
        $b_D \leftarrow k \cdot \big(1/(n-1)\big)$
        $b_T \leftarrow i$
return $b_{min}, b_D, b_T$
```

first shader (Figure 4.7) and finds the best two matches (two lowest distance values) in each row. These serve as the anchor points. Each object in the database has two anchor points selected which best match two feature points on the target image. This is utilized to obtain a scale factor and rotation estimate for the target image against each object in the database.

If the situation arises that two distinct points on the screen happen to best match the same point in an object in the database the shader assigns default values ($s = 1, \varphi = 0$) since it cannot correctly determine the scale and rotation for that object. In other words, if $x_{1_D}, y_{1_D} = x_{2_D}, y_{2_D}$ (refer to Figure 3.6) the scale and rotation estimates are not able to be calculated and we assume no scale or rotation. If this is not the case, from Equation 3.4 and 3.7 we obtain scale and rotation estimates based on the anchor points. Thus for m objects in the database, we obtain m scale and rotation values. Again, m number of scale and rotation values are each computed in parallel

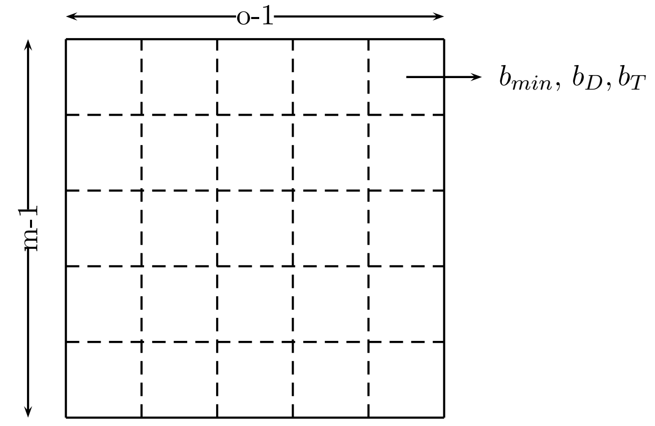


Figure 4.7: Output texture for 1st render pass

	d_0	d_1	d_2	$\cdots$	d_{n-1}
D_0	0.585,0.084	0.533,0.213	0.812,0.331	$\cdots$	0.506,0.777
D_1	0.516,0.063	0.332,0.247	0.599,0.812	$\cdots$	0.724,0.490
D_2	0.857,0.733	0.814,0.836	0.869,0.265	$\cdots$	0.690,0.547
D_3	0.660,0.559	0.700,0.476	0.507,0.688	$\cdots$	0.760,0.209
D_4	0.62,0.572	0.278,0.630	0.4,0.604	$\cdots$	0.420,0.625
D_5	0.541,0.433	0.642,0.290	0.367,0.154	$\cdots$	0.526,0.345
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
D_m	0.533,0.202	0.696,0.475	0.579,0.709	$\cdots$	0.624,0.269

Table 4.4: x, y texture for the database where each row has the corresponds to one database object

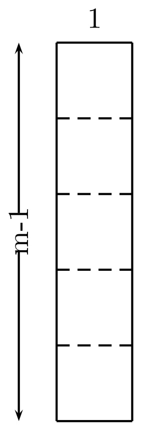


Figure 4.8: Geometry for 2nd render pass

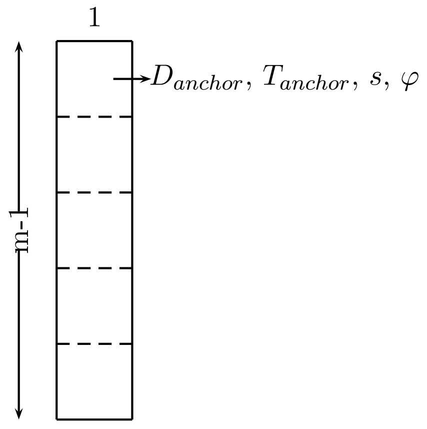


Figure 4.9: Output for 2nd render pass

on separate shader instances.

4.4.3 Shader 3: Feature Point Matching

Using the scale and rotation factor for each of the objects in the database, this render pass transforms each point in the target object with respect to the first anchor point for that database object. It then calculates an overall match value for each object in the database.

This render pass takes the following textures as input:

1. the $RGB\theta$ feature vector in T (see Table 4.1)
2. the $RGB\theta$ feature vector for each object in D (see Table 4.2)
3. the x, y feature vector in T (see Table 4.3)
4. the x, y feature vector for each object in D (see Table 4.4)
5. The output from the previous render pass (see Table 4.9)

The geometry passed in is the same as that passed into shader 2 (see Figure 4.8).

For m objects we obtain m number of scale and rotation values. The target object is transformed m times and matched against the corresponding database object. This render pass starts by looking up the scale and rotation factor for the given object in the database. The next step is to look up (in the output from shader 2) the first anchor point for that object and the corresponding point on the target image. This is used to transform each point t_i on the screen. Again, each of these m transformations and match calculations are done in parallel.

The scale factor obtained is clamped between 1/16 and 16 as scales lower and higher than those values are unlikely. The following formula is used:

$$x'_i = x_{1_D} + s \cdot \Big(\big(\cos\varphi \cdot (x_{i_t} - x_{1_T})\big) - \big(\sin\varphi \cdot (y_{i_t} - y_{1_T})\big) \Big) \tag{4.4}$$

$$y'_i = y_{1_D} + s \cdot \Big(\big(\sin\varphi \cdot (y_{i_t} - x_{1_T})\big) + \big(\cos\varphi \cdot (y_{i_t} - y_{1_T})\big) \Big) \tag{4.5}$$

where (x', y') are the transformed coordinates of the original point t_i. $(x1_D, y1_D)$ is the first anchor point in the database and $(x1_T, y1_T)$ is the first anchor point in the target. The variables s and φ are the scale and rotation factor looked up in the texture. This gives us a predicted location for each feature point. This is used along with the $RGB\theta$ values to perform matching. The distance between the $RGB\theta$ values of every point in the target image with every point in the object summed with the distance between the x', y' values and those in the object gives a match value M_i. These two distance values are weighted before summing so that each measure affects the overall results differently. The ideal weighting w was found to be 50% $RGB\theta$ and 50% x, y values. This value was experimentally chosen and the method for arriving at this value is described in the Results section.

$$M_{i_{xy}} = d\big(d_{i_{xy}}, (x_i', y_i')\big) \tag{4.6}$$

$$M_{i_{rgb\theta}} = d(d_{i_{rgb\theta}}, t_{i_{rgb\theta}}) \tag{4.7}$$

$$M_i = (1 - w) \cdot M_{i_{xy}} + w \cdot M_{i_{rgb\theta}} \tag{4.8}$$

The next step is to create a bias towards matching objects in the database that were found to have a scale factor near 1 (no scaling) and a rotation near 0 degrees (not rotated) against the target as it is more likely to be that object. During testing it was noticed that when using a low number of feature points, there was a high possibility of obtaining false positive matches due to random objects at various scale and rotation combinations matching the target image by chance. In order to avoid this situation, a bias is introduced. The formulae below give the bias values for the scale and rotation that were chosen in this thesis:

$$b_s = \frac{|\log_2 s|}{4.0} \quad 0 \le b_s \le 1 \tag{4.9}$$

$$b_\varphi = 1.0 - 2.0 \cdot |\varphi - 0.5| \quad 0 \le b_\varphi \le 1 \tag{4.10}$$

where the variables s and φ are the scale and rotation factor and b_s and b_φ are the bias values that get added on to the match value. The lower

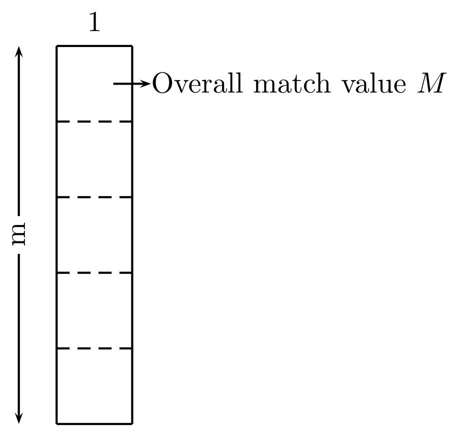


Figure 4.10: Output for 3rd render pass

the match value, the better the match found. Therefore, a lower bias value corresponds to the default scale and rotation of 1 and 0 respectively.

The overall match value computed is placed in a texture and the resulting output texture is passed back to the CPU-side host application which then iterates through the result texture for each object and finds the best five matches.

Most algorithms try to avoid $O(n^2)$ operations where every feature point on the target image must be matched against every feature point in every object in the database. However, since this matching scheme is specifically designed for the GPU which executes multiple instances in parallel this becomes an $O(n)$ operation in time.

4.5 Testing

Several tests were conducted to determine the relative weighting of $RGB\theta$ versus x, y values, scale factor prediction, rotation prediction and feature matching. Each of these tests is described in this section followed by the results for the tests in the next chapter.

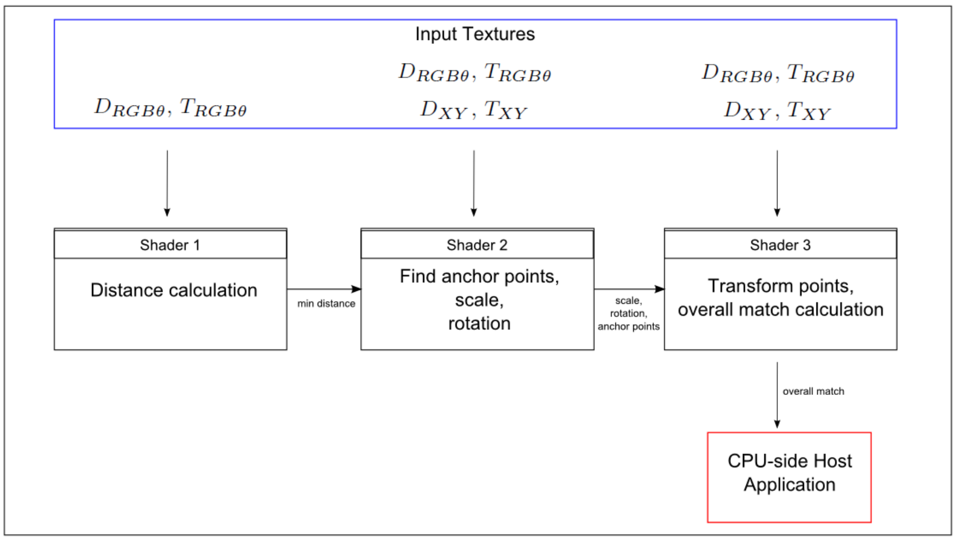

Input Textures
$D_{RGB\theta}, T_{RGB\theta}$
$D_{RGB\theta}, T_{RGB\theta}$ D_{XY}, T_{XY}
$D_{RGB\theta}, T_{RGB\theta}$ D_{XY}, T_{XY}
Shader 1
Distance calculation
min distance
Shader 2
Find anchor points, scale, rotation
scale, rotation, anchor points
Shader 3
Transform points, overall match calculation
overall match
CPU-side Host Application

Figure 4.11: Entire GPU Pipeline for feature point matching using spatial locality

4.5.1 Relative Weighting

The previous feature matching conducted in (Hall, 2014) uses purely colour change to match a feature vector from the target set to a feature vector in the database set. This work extends the feature descriptor to a six-dimensional vector and uses spatial locality in conjunction with a scale and rotation estimator to perform matching. This set of tests aims to select the ideal relative weighting between colour change and spatial information contributing to the final matching result. To accomplish this, a series of tests were performed on a database of 50 common company logos (see Figure 4.3 on page 52). The feature matching results were recorded for each of the 50 logos and this test was repeated twice giving 100 results. During each iteration of the test, the matching was performed with different relative weights starting from 10% $RGB\theta$ versus 90% x, y up to 100% $RGB\theta$ versus 0% x, y with a 10% increment between each test. These results were recorded and analysed for each weight combination and the combination selected that gave optimum matching.

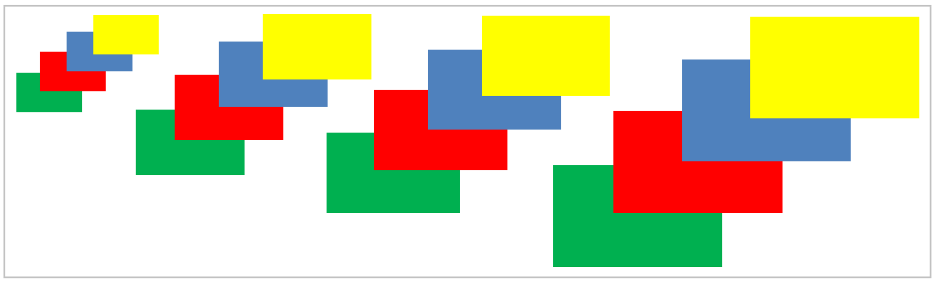


Figure 4.12: Same object at different scales

4.5.2 Scale and Rotation Estimation

The next test performed measured the estimation accuracy of the scale and rotation values given a target object. Since the scale and rotation calculations were to be tested in isolation to obtain a measure of their accuracy, the tests were performed on the objects in Figure 4.12 and Figure 4.13. The results of these tests recorded the actual scale or rotation of the object versus the estimated scale or rotation of the object. This was performed at scales ranging from 0.5 to 2. Due to limitations of the camera, very small and very large scales could not be accurately tested although theoretically this method should work for all scales between 1/16 and 16. The rotation testing was performed starting at a 0 degree rotation up to a 350 degree rotation with 10 degree increments.

4.5.3 Feature Matching

Since this is the main purpose of the feature descriptor, it is the most heavily tested. A series of tests were performed as follows:

1. Against the logo database - with no scale, rotation or translation and via manual feature discovery

2. Against a database of famous art work - with no scale, rotation or translation via automated feature discovery

3. Against a database of famous art work - including translation via automated feature discovery

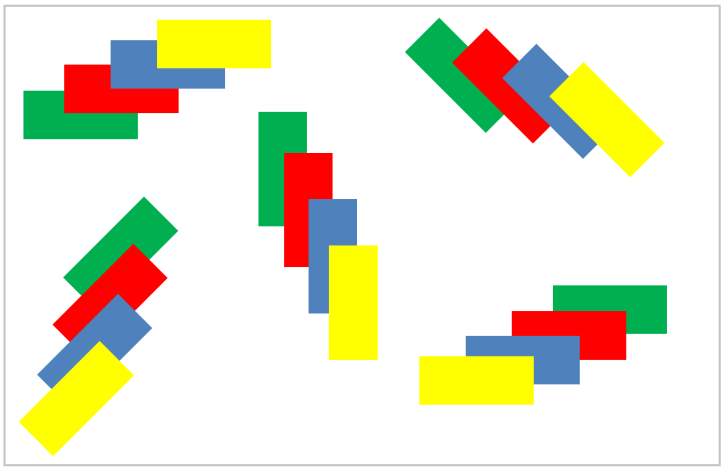


Figure 4.13: Same object at different degrees of rotation

4. Against a database of famous art work - including scale and rotation changes via automated feature discovery

The first phase of testing involved feature point extraction from each logo using the contour tracking algorithm. Feature points are extracted by first clicking on one edge of the logo and finding further feature points along the contour that are local maximum. The next phase of testing involved recognizing each logo against the 50 logos in the database and recording the results of the algorithm with and without the x, y spatial locality information. There are two accuracy measures being investigated. The first was whether or not the correct logo was identified by the algorithm and the second is the difference between the logo identified by the algorithm as its first and second prediction (as a measure of how distinctive the match was). Let M be the measure being recorded. If M_0, M_1, M_2, M_3, M_4 give the overall match value for the closest five matches between the target image and the objects in the database and O_0, O_1, O_2, O_3, O_4 be the corresponding objects picked as the first to fifth best match respectively:

$$M = \begin{cases} M_1 - M_0 & \text{if } i = 0 \\ M_0 - M_i & \text{if } 1 \leq i \leq 4 \\ -127 & \text{if } i \geq 5 \end{cases}$$

where i is the position at which the correct object is picked. This gives us a conclusive measure of how accurately the feature matching algorithm performed for a given test. This measure shall be used for all the feature matching tests performed.

The second set of tests starts by drawing a grid around the object of interest and finding all the points within that grid that are above a certain strength threshold. We begin with a grid of size 1100×620 which amounts to 682,000 feature points. Using a threshold feature points that do not meet our strength criteria are eliminated. This gives us approximately between 10,000 to 200,000 feature points depending on the image. After sorting these points a distance threshold is used to select features that are a certain distance away from each other.

To establish the merits of spatial locality being included in the object recognition process we created and used a database of famous art works as they exhibit a wide range of colour changes and are more realistic objects than computer generated logos. Art work also makes good use of spatial information as the relative location of features points stay the same between different images of the same painting. It is particularly difficult to find standard databases to perform object recognition tests that do not presume the use of a high level machine learning algorithm. Moreover, the machine learning databases contain images whose resolution and quality are insufficient for our tests. The objects within these databases contain significant variation within the same category which does not suit our low-level technique of feature point matching where there is no training or learning involved and all of the matching is performed on the device in real-time. For example, the two objects in Figure 4.14 are obtained from the de facto machine learning database (Fei-Fei, Fergus, & Perona, 2004). Both these objects

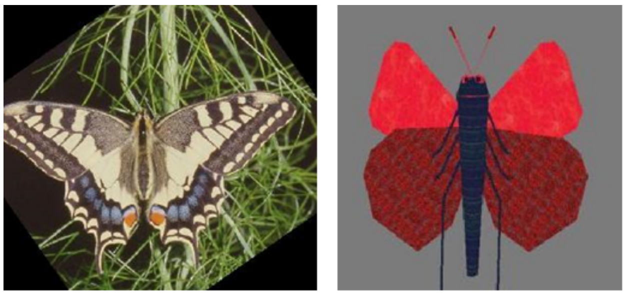


Figure 4.14: Two objects from the same category in de facto ML database (Fei-Fei et al., 2004)

would be classified as the same even though they are vastly different colours and shapes. This sort of categorisation does not work with our approach to feature matching as this would require sophisticated machine learning techniques to be applied. The aim is that the feature descriptor proposed here be used within other high level machine learning classifier functions. As a result, we built our own custom database consisting of art work and tested the algorithm against images of such paintings obtained from the web.

The test set and reference set use different images of the same painting (refer to Figure 4.15). The images have slight variation in colour, scale and aspect ratio. These tests were conducted using 100 paintings in the database tested against different images of the 100 paintings using different scales, rotations and view points.

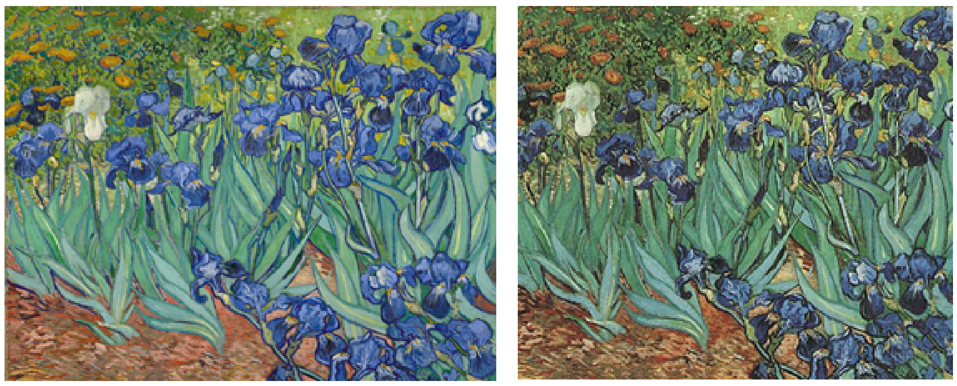


Figure 4.15: An example image from the reference set (left) and the test set (right): Irises, Vincent van Gogh (1889)

Chapter 5

Results and Analysis

All the tests in this section were conducted on the Samsung Galaxy S4 GT-I9505 with an Adreno 320 GPU running the Android operating system. The resolution used is 1280×720 and each test uses the in-built device camera against an image captured on a desktop screen. As a result, the tests are intentionally prone to changes in viewing conditions such as skew, scale, rotation, lighting changes and glare on the screen affecting the colour.

5.1 Extending ColourFAST Descriptor

The first test conducted aims at determining whether using spatial locality along with the original four-dimensional ColourFAST feature descriptor was beneficial to the matching process. This is done by performing a series of 100 tests against a database of 50 logos. Each logo has its feature points extracted one at a time and stored in a database. Typically, each logo contains 15 to 20 feature points. Every logo in the test set is matched against the reference set and the difference measure M is recorded. Let M_0, M_1, M_2, M_3, M_4 give the overall match value for the best five matches between the target image and the objects in the database and O_0, O_1, O_2, O_3, O_4 be the corresponding objects picked as the first to fifth best match respectively,

$$M = \begin{cases} M_1 - M_0 & \text{if } i = 0 \\ M_0 - M_i & \text{if } 1 \leq i \leq 4 \\ -127 & \text{if } i \geq 5 \end{cases}$$

where i is the position at which the correct object is picked. The greater the value of M is, the better the match since that indicates a more definite match. The value of M is clamped to -127 as a larger negative value than that is unlikely in practice. Therefore, $M \in [-127, 255]$ where -127 is assigned to M when the correct object is not found within the best five matches and 255 is assigned to M when the correct object is picked first with an overall match value of 0 and the other top four matches have an overall match value of 255. However, for an overall match of 0 every feature point on the target image must exactly match some point within the object in the database for all six $RGB\theta$, xy components which would be highly improbable in practice. For this set of tests the highest value obtained is 79 and the lowest is -127 (results below -100 are not plotted as the actual match value is unknown when the correct object is not picked within the first five matches and as such is assigned the value -127 to penalize it). Note that the values M_0, M_1, M_2, M_3, M_4 are based on a modified Euclidean distance calculation and therefore the positive values closer to zero are better matches. Whereas the value of M is the difference between two of these values and, therefore, the greater values indicate higher match accuracy.

The box and whisker plot (Figure 5.1) shows ten bars, one for each of the combinations. Recall the Equation 4.8 on page 62:

$$M_i = (1 - w) \cdot M_{i_{xy}} + w \cdot M_{i_{rgb\theta}} \tag{5.1}$$

The horizontal axis represents the chosen value of w which controls the contribution of $M_{i_{rgb\theta}}$ versus $M_{i_{xy}}$ to the overall match value. The plot shows the median, lower quartile, upper quartile and outlier values for each of the combinations. From the results it can be seen that $w = 0.5$ has the highest median value, however, $w = 0.6$ and $w = 0.7$ both have higher upper quartiles. Each of these w values yield the best match accuracy and their results differ marginally. This is, therefore, the ideal combination of $RGB\theta$ and x, y values to be used. As w is a parameter that is fed in to the GPU pipeline by the CPU-side host application, it can be tweaked and modified depending on the requirements of the test being conducted.

It is clear from this set of testing that the addition of spatial locality benefits the feature matching process as the results where $w > 0.7$ (x, y contribute less) yield progressively lower median and lower quartile values. When $w < 0.3$ ($RGB\theta$ contribute less) the median and lower quartile values are significantly worse. This indicates that the x, y and colour change values perform worse when used in isolation for feature matching; on the other hand they improve the accuracy of matches when used in conjunction.

For all further tests a combination of 50% $RGB\theta$ and x, y is chosen.

5.2 Feature Point Matching: Preliminary Test

Since the combined use of colour change values and spatial locality were beneficial to the feature matching process, the next set of tests focuses on comparing the accuracy of matches obtained via the extended descriptor versus the original ColourFAST descriptor. The initial set of feature point matching tests were conducted on a database of computer generated logos (shown in Figure 4.3 on page 52). This data set was chosen due to the nature of logos having sharp edges for contour tracking and feature discovery as well as high contrast between the object of interest and the background yielding distinctive $RGB\theta$ values. This was the database used by Hall for testing purposes and, therefore, was the natural data set of choice for preliminary testing. The feature points were extracted using the procedure described in Section 4.2 and each logo is matched against the other 50 logos over a series of 50 tests. The same metric, M, is recorded which gives the accuracy of the match test performed. Figure 5.2 shows the results of this set of tests. The median and lower quartile values are greater for the spatial locality results. However, for both sets of data the upper extreme value and the

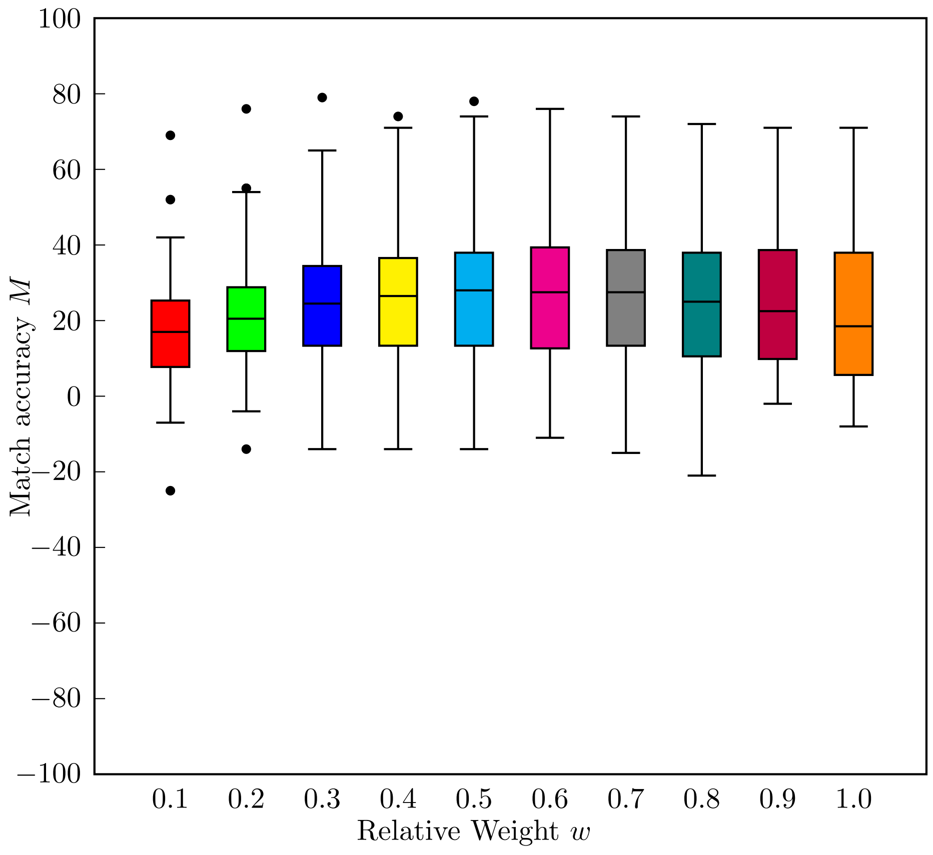


Figure 5.1: Box and whisker plot of matching accuracy M for different relative weight combinations (excluding results below -100)

upper quartile value do not differ substantially. This led to further analysis of the data to obtain a measure of accuracy improvement achieved by adding x, y values.

As the variances of the two sets of data are unequal, a Welch's t-test was conducted on both versions of the descriptor (with and without spatial information) to determine whether or not there was an improvement in feature point matching by extending the descriptor. This test revealed that there was not a statistically significant difference in terms of match accuracy between the two versions for matching computer generated logos as can be seen from Table 5.1. In reality a t-test is not the ideal method of comparing the two sets of data, however, it does show that there is no substantial difference in accuracy as both result sets show a close similarity.

Two-Sample Assuming Unequal Variances		
	Spatial Locality	ColourFAST
Mean	-0.18	-8.78
Variance	1548.232245	2122.379184
Observations	50	50
Hypothesized Mean Difference	0	
df	96	
t Stat	1.003723858	
$P(T <= t)$ one-tail	0.159017508	
t Critical one-tail	1.66088144	
$P(T <= t)$ two-tail	0.318035015	
t Critical two-tail	1.984984312	

Table 5.1: Welch's t-test using samples from Figure 5.2

The insignificant difference between results is due to the nature of the test performed. Each logo being matched against the database was not scaled or rotated and was placed in the same position as during the feature extraction phase. In addition, the test set used was the same as the reference set and, thus, did not produce much x, y movement. This resulted in the scale and rotation descriptor not being utilized ($s = 1$ and $\varphi = 0$ by default) and, hence, the x, y values are not greatly transformed. Since the target image was not significantly transformed with respect to the database set,

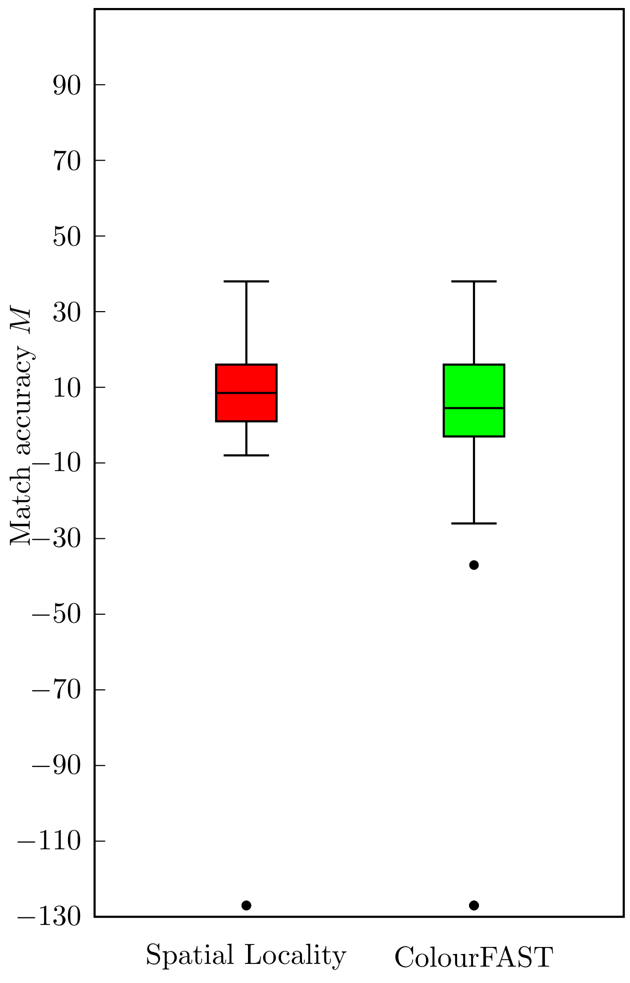


Figure 5.2: Boxplot of feature point matching accuracy for the extended descriptor versus the original ColourFAST descriptor for logos

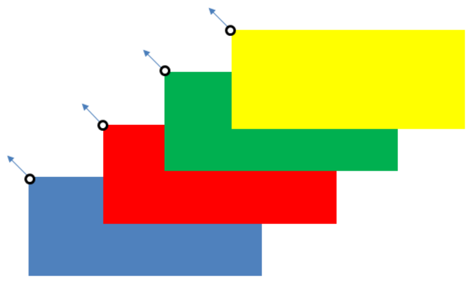


Figure 5.3: Same object at different scales

the x, y values did not improve the matching accuracy. Even though this test uses ideal, simplified objects and, therefore, does not rigorously tests the accuracy of the descriptor it has been used here as it is the same test conducted in (Hall, 2014) and serves as a useful preliminary indicator of whether the spatial locality information is beneficial before performing tests on more complex objects.

5.3 Scale and Rotation Estimation

This set of tests conducted evaluate the accuracy of the scale and rotation estimates. Each of these were tested in isolation; first rotating the object between 0 and 350 degrees with a 10 degree increment and then scaling the object between 0.5 and 2. Each of the predictions were recorded alongside the actual scale and rotation of the object and the error margin for each test is plotted. In order to purely test the scale and rotation, a simplistic object with four feature points was picked as shown in Figure 5.3.

Figure 5.4 shows a scatter plot for the rotation estimate tests. The horizontal axis shows the actual rotation of the target image and the vertical axis shows the estimated rotation calculated. The results of four samples each containing 36 tests ranging from a 0° rotation to a 350° rotation with a 10° increment are plotted. Each set of tests has been plotted using a different colour to indicate that the test has been repeated several times. Using regression analysis on the data obtained the trendline is given by $y = 0.9983x - 2.0369$ which gives the best-fit straight line for the data set.

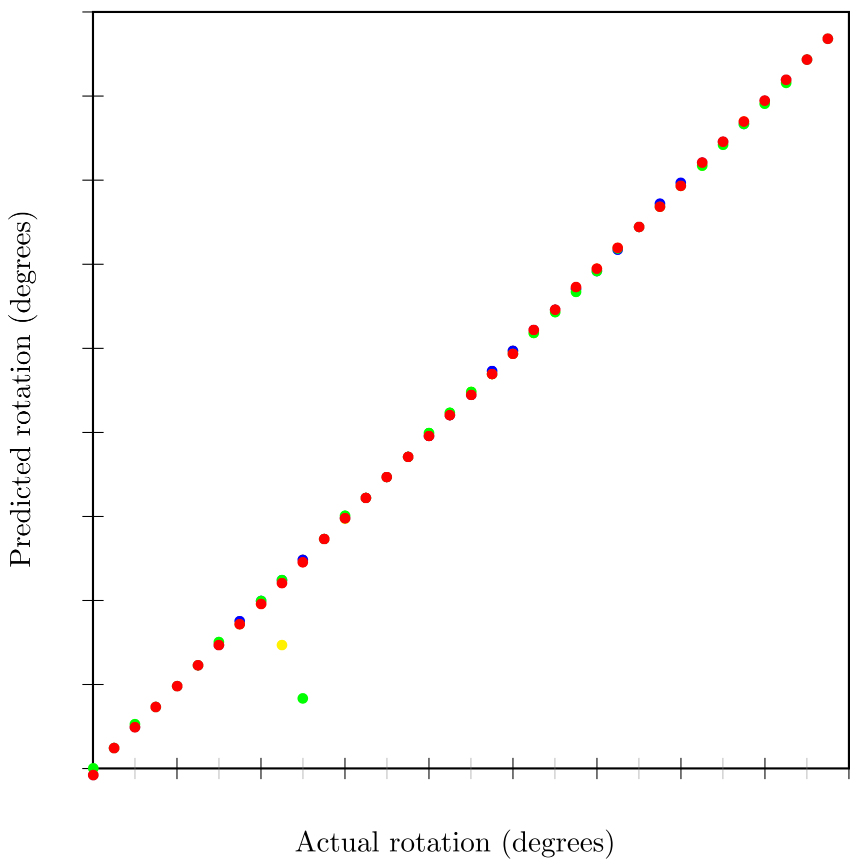


Figure 5.4: Actual rotation versus predicted rotation

The R-squared value obtained for the data is 0.9967. This illustrates the very high accuracy achieved by the rotation predictor.

Figure 5.5 shows a scatter plot for the scale estimate tests. The horizontal axis shows the actual scale of the target image and the vertical axis shows the estimated scale obtained. The results of nine samples each containing 16 tests ranging from a scale factor of 0.5 to 2.0 with an increment of 0.1 between tests are plotted. Using regression analysis on the data obtained the trendline is given by $y = 1.0085x - 0.0084$ which gives the best-fit straight line for the data set. The R-squared value obtained for the data is 0.9761

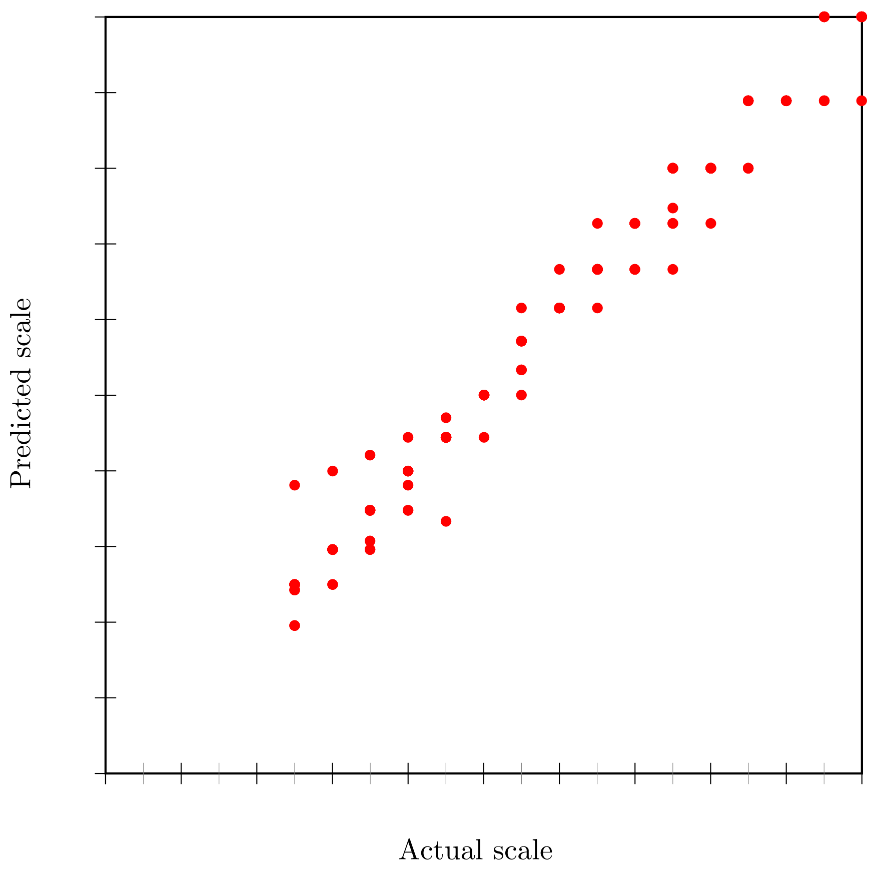


Figure 5.5: Actual scale versus predicted scale

which again indicates the good accuracy achieved by the scale estimation calculation.

The scale and rotation estimates are based on the formulae discussed in Section 3.3 which uses two anchor points as the basis to obtain this estimate. From the results above, it is evident that for simplistic objects with ideal feature point vectors, as in the object used here, the scale and rotation values determined are particularly accurate.

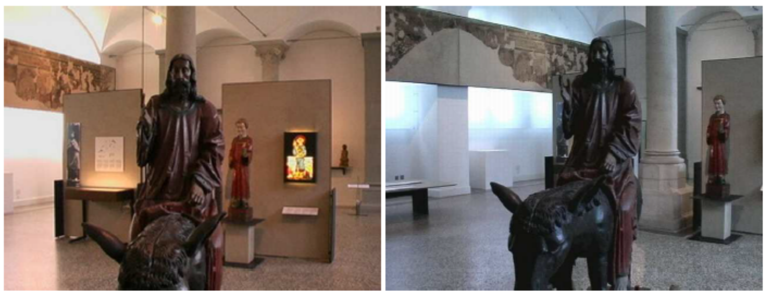


Figure 5.6: An example image from the reference set (left) and the test set (right): SURF
(Bay et al., 2006)

5.4 Feature Matching: Secondary Test

This set of tests attempts to measure the feature point matching accuracy on real-world objects (as opposed to the previous test which utilized computer generated logos). SURF uses a database of 216 images of 22 objects of art in a museum, on average using approximately nine images under different conditions for each object. The test set consists of 116 images and are taken in various conditions such as extreme lighting changes, objects reflecting in glass cabinets, changes in viewpoint, zoom and differing camera qualities (for example, refer to Figure 5.6). Due to the lack of standard databases available for such tests conducted on a mobile phone, the database used had to be created for this specific purpose (shown in Figure 5.7). We attempted to model our database after the approach used by SURF. It consists of 100 images of famous paintings, one image for each painting, all taken via a mobile phone camera facing a desktop screen containing a picture of the painting. The test set is a subset of these paintings containing 50 images obtained from different sources than the reference set. Thus, each of these paintings differ slightly in terms of aspect ratio, colour and resolution (as can be seen in Figure 4.15 on page 69).

The evaluation metric, M, used is the same as that used in the preliminary tests carried out against the logo reference set. However, the feature discovery and extraction phase is fully automated for these tests as described

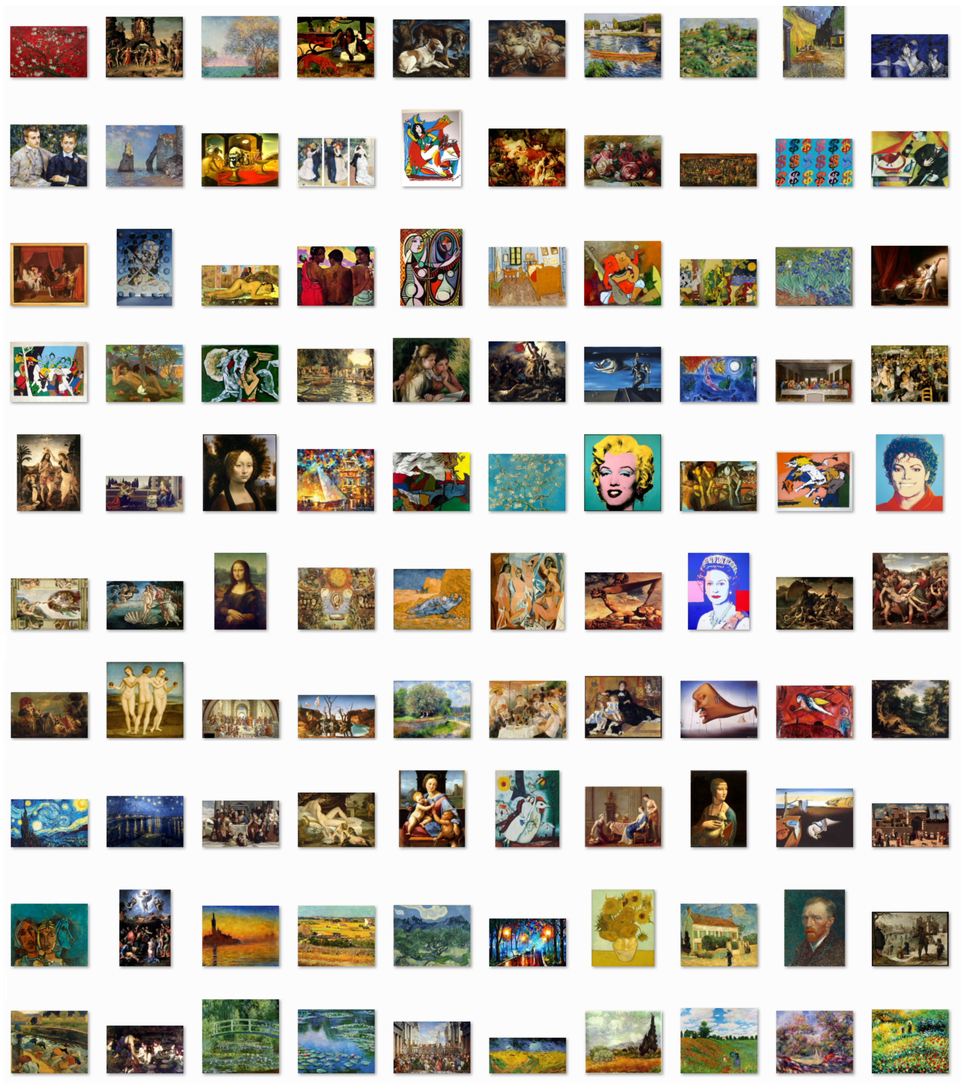


Figure 5.7: Database used for the second series of feature matching tests

in Section 4.3 and, hence, allows for more realistic and rigorous testing. The art work reference set provides a set of complex and intricate real-world objects to test against. The box and whisker plot in Figure 5.8 shows the accuracy of the matches between the descriptor using spatial locality and the original ColourFAST descriptor. This series of 50 tests were performed under minor scale, rotation, skew and view point changes to purely test the matching capabilities of the feature points. Although, since the feature discovery is automated it is not guaranteed to pick the same feature points as is in the reference set for a given object. Similar to the first test, the greater the value of M, the better the match accuracy obtained. $M \in [-127, 255]$ and the maximum value for the spatial locality data sample is 46 compared to the maximum value for the original ColourFAST data sample which is 14. The median values are 5 and 0.5 respectively and the lower and upper quartile values are significantly greater for the extended descriptor. The sizeable drop in match accuracy for ColourFAST can be attributed to the fact that using the 4-dimensional vector alone is not distinctive enough on objects such as paintings that do not have sharp, defined edges. Logos typically have between two and five colours with sharp corners and edges against a white background making the colour change values sufficiently unique. The reference set containing paintings presents a much more challenging task in terms of matching feature vectors. In addition, the automated feature discovery leads to different features being picked from those extracted and saved in the database. Thus, the benefit of using spatial locality in conjunction with colour change values and correcting for changes in scale, view point and rotation exhibits itself more clearly in this set of tests.

Once again, a Welch's t-test was conducted on both data sets as the two samples have unequal variance. This is used to determine whether there is a statistically significant difference in accuracy between the data samples. In this situation, there is a considerable difference between the values of M for ColourFAST feature points versus the extended version that includes spatial locality information as shown in the results from the t-test in Table 5.2.

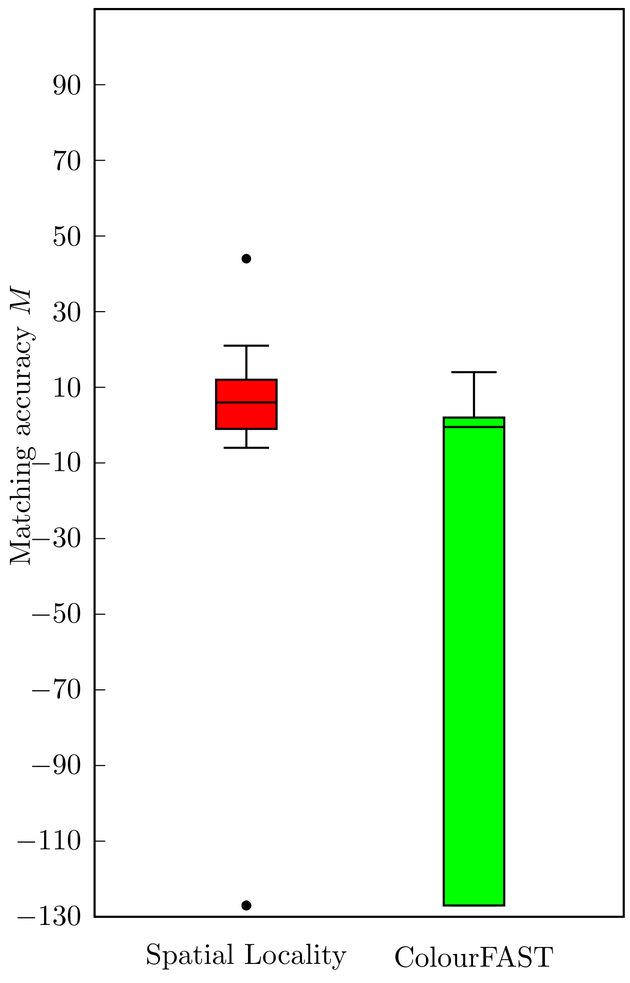


Figure 5.8: Boxplot of feature point matching accuracy for the extended descriptor versus the original ColourFAST descriptor for paintings

Two-Sample Assuming Unequal Variances		
	Spatial Locality	ColourFAST
Mean	-10.38	-49.64
Variance	2353.873061	4081.7453
Observations	50	50
Hypothesized Mean Difference	0	
df	91	
t Stat	-3.46051	
$P(T <= t)$ one-tail	0.0004114	
t Critical one-tail	1.6617712	
$P(T <= t)$ two-tail	0.0008228	
t Critical two-tail	1.9863772	

Table 5.2: Welch's t-test using samples from Figure 5.8

5.4.1 Feature Matching: With Scale and Rotation

Next, the same test set was used to test the match accuracy under different rotations starting with a 10° rotation up to a 30° rotation with 5° increments. To eliminate the border entering the search grid upon rotation the target image used is slightly zoomed in. The results are shown in Figure 5.9. The median values for the two samples are -3 and -1.5 respectively; the upper quartile values are 5 and 2.75 with the spatial locality sample marginally outperforming the other. However, the match accuracy is shown to have dropped tremendously with the lower quartile values for both samples at -127. It is clear from the t-test results in Table 5.3 that with the target object scaled and rotated the matching accuracy severely decreases and the x, y values no longer significantly benefit the feature point matching process. This unsuccessful result is expected since if the anchor points $(x1_T, y1_T)$, $(x1_D, y1_D)$ and $(x2_T, y2_T)$, $(x2_D, y2_D)$ are not indeed the same corresponding points as is assumed, the scale and rotation calculations obtained are incorrect. This in turn affects the transformation of each point on the target image causing the accuracy to decrease significantly. This has been noted as a limitation of our approach of taking only two anchor points to calculate the scale and rotation. This is also often caused due to the method of testing because a major portion of the image is lost when the target object is rotated. Consequently, a subset of feature points originally extracted for the reference set are lost (illustrated in figures 5.11, 5.12 and 5.13). As a result, the chances of the anchor points being picked incorrectly increase.

Two-Sample Assuming Unequal Variances		
	Spatial Locality	ColourFAST
Mean	-53.16	-54.52
Variance	4388.341224	4217.315918
Observations	50	50
Hypothesized Mean Difference	0	
df	98	
t Stat	-0.10366494	
$P(T <= t)$ one-tail	0.458823551	
t Critical one-tail	1.660551217	
$P(T <= t)$ two-tail	0.917647103	
t Critical two-tail	1.984467455	

Table 5.3: Welch's t-test using samples from Figure 5.9

5.5 Discussion

Comparing the results of the tests performed against the two reference sets (i.e. logos and paintings) it is clear that the spatial locality information had a negligible effect when using the logos as test subjects. This is due to several reasons. Firstly, the feature points were manually picked and extracted on the logos before saving to the database. During the feature matching process, each feature point was manually placed on the screen. This ensured that roughly the same feature points were chosen during the extraction and matching phase which resulted in minimal x, y movement for the feature points. Moreover, the choice of feature points was possibly unambiguous due to the RGB values being so similar.

Let $(x1_T, y1_T)$, $(x2_T, y2_T)$ and $(x1_D, y1_D)$, $(x2_D, y2_D)$ be the anchor points picked on the target and reference object respectively. If $(x1_D, y1_D) =$

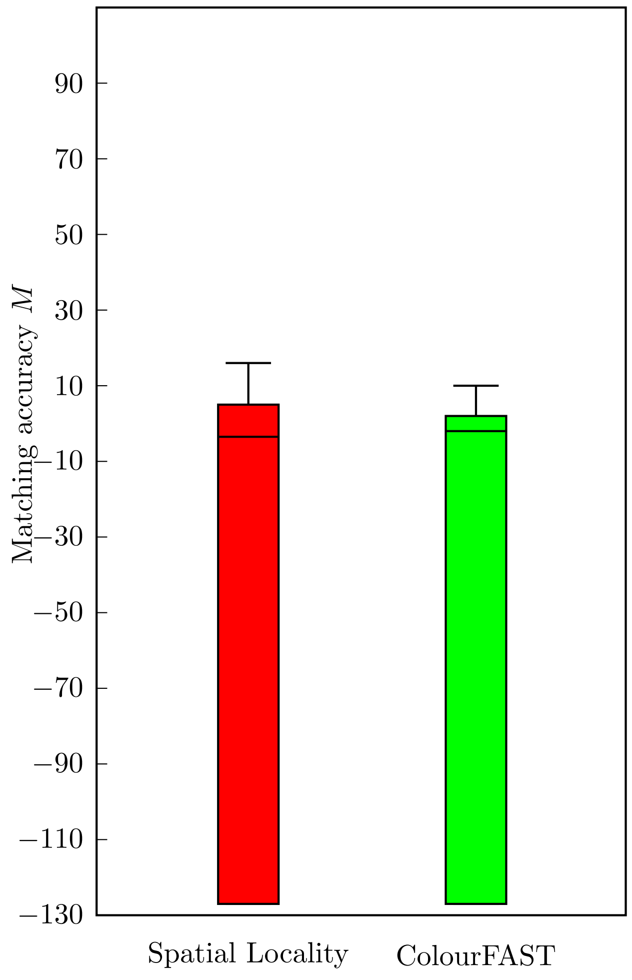


Figure 5.9: Boxplot of feature point matching accuracy for the extended descriptor versus the original ColourFAST descriptor for paintings with rotation

(x_{2_D}, y_{2_D}) we cannot gain a sense of scale or rotation resulting in the default values $(s = 1, \varphi = 0)$ being used. This is typically the case with logos as they tend to be two-toned images where several feature points within the image have the same colour change values with only the θ component differing. As an example, refer to Figure 5.10 of a common logo where every feature point is identical in terms of colour change. As the θ component is weighted lower in the anchor point picking phase, this results in the same feature point within the reference object best matching several different feature points within the target image. This is described in Section 4.4.2 on page 57. When this situation arises, the transformation of each target feature point is very small leading to the x, y values being essentially unused.

Another reason for the change in accuracy is the number of feature points used. The logo tests used a relatively small number of points (between 15 and 20 feature points per logo). On the other hand, the tests conducted using the paintings used 60 feature points for each object and due to the paintings being more realistic it is very unlikely that two feature points in the target image happen to best match the exact same point in the database. These tests utilized automated feature discovery which allowed the two data samples to differ significantly. Therefore, the second set of testing yielded more promising results.

The next series of tests were undertaken with the target object rotated and slightly scaled using the paintings as the reference set. The accuracy suffered tremendously during this test with the object not being correctly recognized on more than a quarter of the tests. This is because of the nature of the tests performed. Using a hand-held camera introduces unwanted image deformations such as skew. As the feature point matching is tested on a real-time video feed, the images are not loaded directly into the pipeline for processing but rather obtained via the camera. Figure 5.11 shows a target object initially as its extracted feature points are saved in the database. After rotating the image 20 degrees (Figure 5.12), we lose a major portion of the image and now parts of the border of the image are included in the search grid. This results in the feature discovery phase picking up parts of the border as feature points. To avoid this, the target image must be

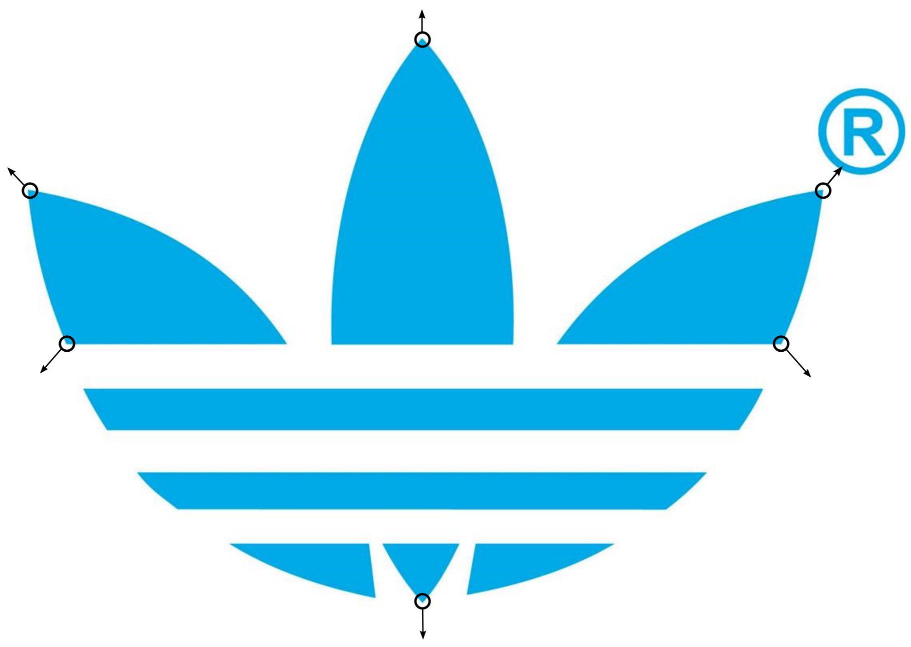


Figure 5.10: Typical logo consisting of several feature points with the same RGB values (only the θ component differing)

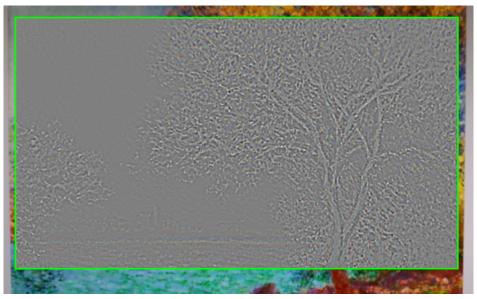


Figure 5.11: Target object used for feature point extraction with no rotation or scaling for saving to database

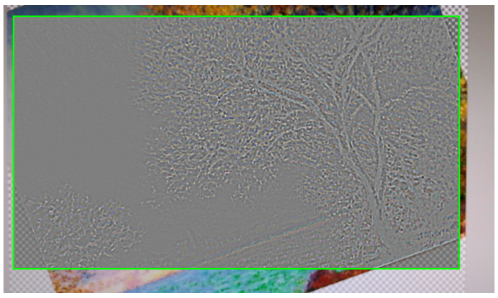


Figure 5.12: Target object after being rotated; search grid now contains parts of the border which negatively affects the feature discovery

zoomed in to eliminate the border as shown in Figure 5.13. After both of these transformations to the object, a large portion of the image is lost and subsequently feature points from the original image are missing. This leads to a significant drop in match accuracy as the target image has only a small subset of the original feature points saved. Another issue that was noticed was the lack of discriminative power of the descriptor that led to incorrect anchor points being chosen which in turn gives an incorrect scale and rotation factor. Transforming a point based on incorrect estimates gives inaccurate “corrected” x, y values.

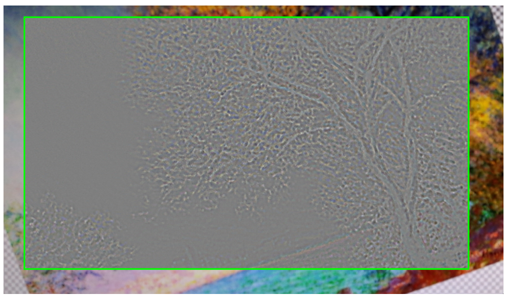


Figure 5.13: Target object after zooming in to eliminate the border; major portion of the image is lost

5.6 Anchor Point Selection

The first test assessing the scale and rotation estimation capabilities of the algorithm used a simple object with four anchor points each with a distinct colour against a white background. Following this, the same test was carried out replacing the target object with a more complex real-world object such as a painting. The estimates obtained were highly inaccurate as opposed to the near perfect results achieved using the square with four feature points. This led to analysing the raw data obtained from the feature discovery phase alongside the data in the database for a given reference object. The painting used as the reference and target object for this analysis is shown in Figure 5.14 and the data used are shown in Appendix A. The data presented in Appendix A model the anchor point selection process performed by the GPU during the first and second render passes of the pipeline. Let $T = \langle t_0, t_1, \cdots, t_{59} \rangle$ be the feature points of the target image and $D = \langle d_0, d_1, \cdots, d_{59} \rangle$ be the feature points of the reference object in the database. The table shows the values obtained by performing the modified Euclidean distance calculation from Equation 3.3 on page 45 for every pair of points. The numbers in bold are the minimum distance value obtained within each instance of the shader code. The next shader then iterates through each of these values and finds two sets of points that correspond to the two lowest distance values to be used as anchor points. From the data it is evident that two feature points from the target image might best match the same point within the reference image. For instance, t_2, t_9 and t_{10} best match the same point d_{11}. Upon further analysis, several limitations of the feature matching approach and method of testing were revealed. Firstly, when no scale and rotation were applied to the target image it was observed that the anchor points (x_{1_T}, y_{1_T}), (x_{1_D}, y_{1_D}) and (x_{2_T}, y_{2_T}), (x_{2_D}, y_{2_D}) were sometimes not in fact the same point contrary to what is expected. This causes incorrect scale and rotation estimates and subsequently incorrect transformations. Secondly, having a two to one mapping between points on the target image and that in the database voids the scale and rotation calculations. This indicates that the anchor points might not always be sufficiently unique for the purpose of recognizing and matching real-world objects. Another challenge faced was successfully testing the rotation and scale invariance. As previously discussed, as the target object is rotated, progressively larger portions of the image are lost. Thus, a testing method is required where rotating the image does not lead to losing a large subset of feature points. A possible solution could be blurring the border of the painting so that upon rotation, the inclusion of the border in the search grid would not have an adverse effect on the feature discovery.

The current approach to feature matching requires that each $t_i \in T$ matches some $d_j \in D$ which is not practical as the target image can be missing certain points due to changes in view point, rotations, skew etc. Thus, it might be more pragmatic to pick a subset of feature points within the target object that must match the reference object feature points. It was hypothesized that stronger feature points lead to better match results. To test this, using the modelling data the features in T and D were sorted based on their feature strength and then matched. Surprisingly, this was not the case in this particular example; with the closer matches found to be towards to the end of the spectrum which correspond to the weaker feature points. Based on the data collected, there seems to be no apparent way to pick a subset of feature points for the target object that yields better matches than using the same number of points in T and D.

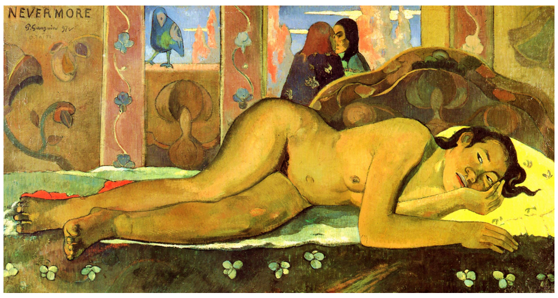


Figure 5.14: Painting used for anchor point selection analysis and modelling: Nevermore, Paul Gauguin (1897)

5.7 Pipeline Performance Analysis

A major focus of this work is designing and implementing a GPU-accelerated feature matching algorithm. The throughput achieved during the feature matching tests conducted in (Hall, 2014) look very promising for real-time object recognition on the GPU. Therefore, it is important to investigate whether the addition of spatial locality is worth the potential decrease in throughput. The algorithms implemented in this thesis have been specifically architected for the GPU pipeline and hence, the trade-off between the match accuracy gained through spatial locality versus the throughput achieved is particularly important to investigate. Hall performs a performance analysis test using the original ColourFAST feature descriptor running the feature matching process each frame against 50 objects in the database and with a varying number of randomized feature points on the target image. We perform the exact same test to compare the frame rate results of the entire ColourFAST feature matching pipeline using spatial locality to the results reported by Hall. This is to ensure that the performance penalty incurred by extending the descriptor does not outweigh the benefits of spatial locality in the matching process. The tests are performed using the same device and screen resolution (i.e. Samsung Galaxy S4 with a resolution of 1280 × 720). The entire feature point matching process is run each frame and the frame rates recorded over a span of a few minutes and then averaged. The accuracy of the matches is not under consideration for this test and is, therefore, not recorded. Table 5.4 shows the performance in terms of frames per second after introducing the spatial locality textures into the GPU pipeline. The first row gives the results reported by Hall using the original descriptor. The next two rows give the frames per second measures for the same test first with 50 objects in the database and next with 100 objects in the database. It is clear that the addition of spatial locality information for the matching does not cause a noticeable difference in throughput. It is noteworthy to mention that the matching scheme using spatial locality requires an additional render pass and performs a scale and rotation estimate calculation before transforming the target image points. Moreover, two additional textures holding the x, y values must be passed into the pipeline per render pass. This explains the decrease in frame rates between from the original matching process that uses a 4-dimensional vector.

	Number of Feature Points				
	5	10	20	50	Set Size
ColourFAST	33.56 ± 1.0	29.52 ± 1.1	20.33 ± 0.8	10.67 ± 0.5	50
Spatial locality	22.1 ± 0.5	17.71 ± 0.4	12.04 ± 0.6	6.44 ± 0.6	50
Spatial locality	16.36 ± 0.5	15.12 ± 0.9	9.739 ± 1.27	4.78 ± 0.3	100

Table 5.4: Throughput results

The next set of results eliminates the tracking phase of the pipeline originally used by the ColourFAST descriptor to recognize and track objects using a live video feed from the camera. For purely testing the GPU pipeline throughput for feature point matching, we use a frozen frame and match a set of 60 feature points on the screen against a reference set of 100 objects. The frames rates recorded were 16.37425 ± 2.0 on average over a period of a few minutes. Thus, we can conclude that for feature point matching, the addition of spatial locality is worth the minimal performance hit incurred.

Chapter 6

Conclusion

As the GPU pipeline has moved away from a fully fixed function pipeline, various application domains have been able to harness the power of the GPU and benefit from its architecture. The GPU is particularly advantageous for tasks that rely heavily on data parallelism where computationally expensive tasks are able to execute independently with little or no intercommunication. As a result, image analysis algorithms can easily exploit the nature of GPUs and can achieve real-time results for a number of tasks such as feature detection, extraction and matching. Most current mobiles support OpenGL ES 2.0 and programmable shaders via GLSL which can be used to implement and test GPU-based algorithms. Combining that with the camera capabilities on modern mobile devices, allow for a large variety of applications such as medical imagery, object recognition and mobile augmented reality. There has been active research in this area in the past decade involving image analysis and processing focusing on high frame rate performance.

In image processing and computer vision a feature refers to some region or point within an image that is considered distinctive in some way, such as being on an edge, corner or blob. Features are widely used for motion tracking, object recognition and scene reconstruction. The Features from Accelerated Segment Test (FAST) algorithm is a very popular and particularly efficient algorithm for corner detection. Previous work in (Ensor &

Hall, 2011) demonstrated that modern smartphones can run computer vision algorithms in real-time when the algorithms are implemented specifically for execution on the embedded GPU. More recently, Ensor and Hall have developed a new alternative to the FAST corner detection algorithm, termed ColourFAST, which is specifically designed for GPU pipelining and utilizes colour information to extract features from an image frame. It extracts feature description vectors for each point while maintaining performance at least as good as FAST, which has opened the door for improved computer vision on mobile devices. Their work focuses on using ColourFAST and its feature descriptions for motion tracking. A technique has also been developed for the extraction of ColourFAST feature points which enables a cluster of feature points to be found within an object in a scene. These extracted points could be used for tasks such as tracking and recognition. This was the finishing point of the previously undertaken work and the starting point of this thesis.

The aim of this work was to investigate the use of these feature points to develop a new technique for GPU-accelerated feature matching using spatial locality. The idea is that ColourFAST, being a compact descriptor, could be used within other existing high-level machine learning techniques to train a particular classifier function for object recognition. This was achieved by extending the original ColourFAST descriptor from a 4-dimensional vector containing $RGB\theta$ values to a 6-dimensional vector including x, y values for the feature point. Including this spatial information incurred minimal performance penalty in terms of computing and matching as evidenced by the speed tests conducted. However, it has significantly increased the discriminative power of the feature points in some scenarios as shown in the results section. It has also been able to provide richer information about the target object such as scale and rotation estimates.

This extended descriptor has been implemented in a highly-efficient way utilizing GPU programmable shaders and OpenGL ES 2.0 via GLSL on the Android platform. The algorithm has been tested on the Samsung Galaxy S4 GT-I9505 with an Adreno 320 GPU. It can, however, be ported to any mobile device that supports OpenGL ES. Alternatively, desktop versions could be conveniently implemented using OpenCL or CUDA.

The extension of the descriptor to include spatial locality included several steps. The following were undertaken during this thesis:

1. Automating feature discovery
2. Extracting the feature point coordinates
3. Developing a scale and rotation estimator
4. Developing a feature matching scheme (which includes correcting the scale and rotation of the target via a feature point transformation)

The automated feature discovery allowed testing against a much larger reference set as the feature points no longer required manual selection. It also led to a more rigorous feature point match test as the points selected within the target object are not guaranteed to be the same as the points in the reference object as was the case when using contour tracking. It also facilitated the use of more realistic objects such as paintings rather than logos. Computer generated images such as company logos typically consist of two to five different colours against a white background. Each of the colour combinations used by logos are fairly unique to the particular logo in question and, hence, colour change values alone are able to easily identify the object. When two logos have the same colour combinations, spatial locality becomes a much more important factor. Based on $RGB\theta$ values alone, it is arbitrary which object is identified correctly as seen in Figure 3.2. Using the x, y values of the feature points, we are able to gain an understanding of the object's shape and distinguish objects that have similar colours but distinct shapes. Using complex objects with a wide range of colour combinations clearly showcases the benefit of using spatial locality. In addition, the scale and rotation estimator provides valuable information about the transformation of the target object with respect to the reference object.

A series of experimental tests were designed for measuring accuracy and performance of the original ColourFAST descriptor versus the extended version. These tests were designed to test each of the above phases implemented during this work in isolation and the results are reported in this thesis. The first set of tests aimed to determine the ideal relative weight of $RGB\theta$ values versus x, y values contributing to the overall feature match value M. The match value obtained was recorded for 100 tests and the results were drawn on a box and whisker plot. The results from these tests revealed that approximately 50% colour change values combined with 50% x, y values yield promising results. Thus, using $RGB\theta$ values along with x, y values were more beneficial to the matching process than using each set of values on their own.

The next set of tests that were conducted used the chosen weighting factor for each of the colour changes values and x, y values and evaluated the match accuracy obtained. Logos were used as the first reference set and each logo was matched against the database comparing how accurate the match was using spatial locality versus the original ColourFAST descriptor. The box plot of the results showed that the performance between the two differed marginally with the spatial locality outperforming the original ColourFAST descriptor. However, a t-test conducted on this data showed that this difference in performance was statistically insignificant. This is due to the nature of objects in the reference set. Each logo has reasonably unique colours that set it apart from the rest of the logos. Thus, without using the shape of the object the algorithm was able to identify the logo correctly. The feature points were either selected manually or via contour tracking which led to minimal movement and transformation of the points on the target image. This resulted in the x, y values making a very small difference to the overall matching process. Thus, it was decided that the next series of tests be conducted with a reference set of more realistic objects. For this purpose 100 famous paintings were chosen similar to the reference set utilized by SURF which made use of images of art in a museum. The target set contained 50 of the paintings from the reference set. These were obtained from difference sources than the images in the database and, thus, contain slight variations in colour and aspect ratio. Paintings are a good test subject as they exhibit a wide range of colours. This makes the colour change values alone more ambiguous leading to the spatial locality values contributing more to a successful match. The median values obtained from these tests indicate a substantial increase in match accuracy using spatial locality. A t-test showed that this improvement in accuracy was statistically significant and, therefore, we can conclude that for this set of data spatial locality values provide considerable benefit. This is expected as the paintings used a much larger set of feature points selected via automated feature discovery, introducing a larger amount of feature point movement than during the logo test. Subsequently, the x, y values were of more importance. The scale and rotation estimator were of great benefit as well since they help transform each of the points on the screen and "correct" for any movement or geometric deformations introduced to the target object.

The scale and rotation estimates were first tested on a simplistic object with four feature points as shown in Figure 3.1. These test results yielded very high accuracy for the rotation predictor ($R^2 = 0.9967$) and good accuracy for the scale predictor ($R^2 = 0.9761$). Thus, on simple objects containing a small number of unambiguous feature points the scale and rotation values obtained were remarkably accurate. Next, the same test was conducted replacing the simple square object with the reference set of 100 paintings. This caused the accuracy to drop tremendously demonstrating certain limitations of this approach.

The first limitation is the drop in feature point matching accuracy when the two chosen anchor points are incorrect, leading to an incorrect scale and rotation estimate which causes every point in the target image to be wrongly transformed. This was a situation often encountered when testing the scale and rotation estimates against paintings. Since the paintings provide a set of ambiguous feature points, the two points on the target object and reference object picked as anchors are not the same points as is assumed for the calculation. Modelling performed using raw data indicated that in certain situations, the two anchors points on the target object best match the same point in the reference object causing the scale and rotation variables to be set to their default values. Both these cases cause a sizeable decrease in match accuracy. Thus, for real-world objects using only two points as anchors to gain an estimate for the scale and rotation of an object might be flawed.

Increasing the number of anchor points used to obtain these estimates may create a more robust estimation phase.

The second limitation is the lack of discriminative power due to the compact nature of the descriptor (six dimensions for ColourFAST with spatial locality compared to the 64-dimensional feature vector used by SURF). The feature points were sufficiently unique for the purpose of tracking as the search window used is small and the movements expected between each frame are minimal. When used for feature point matching with a large search window for objects such as paintings, the descriptor proves to be less distinctive as several feature points have similar colour change values within a given neighbourhood. Thus increase in the dimensionality might minimize the ambiguity displayed by the feature points in real-world scenarios. Another option would be using clusters of points while maintaining the compact 6-dimension vector for ColourFAST. Using a cluster of points in a region gives information about the pattern in the neighbourhood. These clusters could collectively be used as anchor points to provide information to perform the required translation of points on the target object.

The last set of tests conducted analyse the throughput achieved with and without the spatial locality information being passed into the pipeline and determine whether or not the performance penalty incurred is worth the increase in match accuracy. For the test set containing 50 objects the performance hit incurred by the addition of spatial information is approximately 0.0615 seconds on average which is minimal compared to the significant boost in match accuracy (as evidenced by the tests conducted against the paintings database).

6.1 Future Work

Due to the size of the ColourFAST feature descriptor, the feature vector extracted may not be sufficiently unique for recognizing real-world objects. As a result, the points picked as anchors during the scale and rotation estimation phase of the algorithm (described in Section 3.3) might be picked incorrectly. If the two anchor points picked between the target image and the database image happen to not be the same point, the scale and rotation factors obtained will in turn be incorrect. This value is used for transforming each point in the target object to match the database object and consequently "correcting" for any geometric deformations (refer to Equation 3.9). To make the process of estimating the scale and rotation more robust a more elaborate scheme could be used where clusters of points are used as anchors instead. With the current approach, if one anchor point is incorrect, the entire matching process suffers. Using a cluster of points would allow for a certain proportion of points picked to be incorrect while still obtaining a fairly true prediction for the scale and rotation. The other issue noted was the low discriminative power of the feature vectors for the purpose of feature matching. While adequate for certain tasks, it proved to be insufficient in some situations for feature matching. Further extending the feature descriptor to include the colour values around the pixel under consideration would give richer information about the pattern in the neighbourhood.

The x, y values are contained in a texture that gets passed into the pipeline for processing. Since only two of the four components in this texture are used currently, the descriptor could be easily extended to add two other components with no modification to the current GPU pipeline or CPU-side host application or set up. Additional values could also conveniently be passed into the pipeline via new textures. The current descriptor is very small compared to most alternative techniques (64 dimensions for SURF, 128 dimensions for SIFT) and still works relatively well for the purpose of matching. Thus, expanding the dimension would make it more distinctive while still remaining compact. Since the dimension of a descriptor has a direct impact on the time taken to extract and match, its size is of particular importance. Alternatively, the equivalent can be achieved by making use of clusters of points. This has the benefit of still using a compact 6-dimensional vector for efficient computing and matching.

Other future work might include porting the automated feature discovery to the GPU via an additional render pass to accelerate the process. This can be done relatively easily by passing in a texture with the feature descriptors for the entire grid to another shader which could compare the descriptors and pick the most distinctive ones based on the selection criteria. So far, this work has tackled geometric transformations of primary importance such as scale and rotation. Other factors such as skew, anisotropic scaling and perspective changes that are of secondary importance would be useful to investigate.

This work focuses on designing and implementing a GPU-based efficient feature matching scheme devised for incorporating spatial locality information into the ColourFAST feature vector. Overall, for real-world objects the spatial locality information clearly benefits the matching process and allows for efficient and accurate feature point matching against a large reference set. For simplistic objects containing a few colours spatial locality does not contribute as much to the matching process as the colour change values alone are adequate to identify the object. However, the match accuracy is in no way hindered by the x, y values. Moreover, the performance penalty incurred by the additional render pass, two extra components added to the descriptor, scale and rotation estimation and transforming the points is minimal compared to the benefit gained in terms of accuracy. The limitation of the approach is using only two points as anchors, although, this can be overcome by using clusters of points which are more distinctive. Therefore, combining the use of spatial locality with colour change values shows potential for feature point matching. This 6-dimensional descriptor could be used within high level machine learning techniques to develop sophisticated object recognition applications.

Appendix A

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}
d_0	2.1858	**0.28**	2.2165	0.8962	0.5817	1.5805	0.5264	1.033	2.1497	2.0495	2.2467	0.9225	0.5582	0.7704	1.8681	1.7728	2.2364	0.5166	1.0065	0.754
d_1	1.9872	0.9181	1.571	0.8996	0.9255	1.8556	1.0103	1.086	1.5787	1.5924	1.5912	0.9748	0.8963	0.8476	2.0244	1.8652	2.0794	0.25	0.9151	0.3
d_2	2.1092	0.6484	2.1159	1.1074	**0.3**	1.7274	0.4505	0.751	2.1371	2.1391	2.141	0.8673	0.29	0.8902	1.8597	1.5921	2.2432	0.8613	0.8106	0.7955
d_3	2.0022	0.9439	1.6682	0.7086	1.1418	1.9857	1.0415	1.1886	1.6557	1.5609	1.5966	1.0151	1.1044	0.7402	1.7909	1.9362	1.8002	0.2546	0.9077	0.3876
d_4	2.0881	0.4893	2.0866	1.1001	0.419	1.5633	0.44	0.908	2.0875	2.042	2.1166	0.9195	0.3946	0.9192	1.8885	1.6156	2.297	0.765	0.9253	0.7419
d_5	**0.34**	2.0919	0.85	1.6128	2.023	0.7961	2.1936	1.8611	0.945	0.5748	0.8065	1.5937	1.9429	1.4506	0.5249	0.46	0.539	1.7562	1.6075	1.5156
d_6	2.2088	0.772	1.9684	1.1521	1.0495	1.7685	1.0594	1.4018	1.8623	1.8437	1.9764	1.2339	1.0245	1.0718	2.0877	1.959	2.2755	0.5084	1.2802	0.7551
d_7	1.0766	1.8144	0.815	2.2069	1.9705	**0.61**	1.986	2.3167	0.7315	0.7756	0.8319	2.1996	1.9458	2.0862	1.0082	0.804	1.2254	1.5616	2.2088	1.6584
d_8	2.2018	0.7273	2.4265	1.1354	0.601	1.8299	0.6843	1.0876	2.3449	2.2272	2.4073	1.0404	0.4611	0.9447	1.7699	1.6804	2.2231	0.9505	1.1355	1.0921
d_9	1.2335	1.9118	1.222	2.2031	1.5774	0.9284	1.5532	1.8574	1.1318	1.3182	1.1688	2.086	1.4022	1.9966	0.6817	0.7044	1.0825	2.0618	1.8848	2.0681
d_{10}	1.5906	1.1631	2.1403	0.7554	0.9894	1.9726	1.2109	0.6134	2.2396	1.8847	2.1056	**0.3**	0.8723	0.4455	1.6025	1.4977	1.697	0.9549	0.4499	0.7311
d_{11}	0.9851	2.1479	**0.48**	1.9565	2.1092	1.0082	2.0702	2.0543	0.56	**0.53**	**0.34**	2.0613	2.087	1.9157	0.9887	0.9532	0.8723	1.5066	1.8103	1.3763
d_{12}	0.5365	2.0389	0.9816	1.3863	2.0326	1.0106	2.1953	1.8537	1.079	0.6381	0.9151	1.5471	1.9561	1.3685	0.724	0.6887	0.5454	1.6328	1.5426	1.4097
d_{13}	1.8919	0.5633	1.8833	1.1009	0.7678	1.3045	0.7987	1.1583	1.8677	1.724	1.9154	1.0335	0.7452	0.9339	1.7263	1.522	2.0589	0.6395	1.1164	0.7327
d_{14}	1.487	1.1442	2.0653	0.63	1.1011	1.9622	1.3025	0.7921	2.1679	1.7504	2.0206	0.4585	1.0002	**0.34**	1.5288	1.5244	1.5096	0.8735	0.4691	0.7246
d_{15}	0.9225	2.2458	1.4274	1.6853	2.1842	1.3464	2.1972	1.6814	1.397	1.1789	1.305	1.385	2.0602	1.4461	0.6635	0.8599	0.665	1.9896	1.4239	1.9025
d_{16}	1.7804	1.0959	1.6277	0.7491	1.2689	1.8703	1.2046	1.2828	1.6143	1.3433	1.4938	1.0637	1.2212	0.7799	1.6283	1.7611	1.5904	0.4559	0.9624	0.4372
d_{17}	0.8737	1.8978	1.0659	1.6854	1.8284	1.0051	1.7404	1.639	1.0166	1.0078	0.985	1.6944	1.7625	1.4953	**0.34**	0.5782	0.7026	1.7113	1.5013	1.7046
d_{18}	1.748	0.9992	1.9673	1.0578	0.663	1.7289	0.8719	0.6585	2.0309	1.9135	1.9722	0.7436	0.5672	0.7848	1.5074	1.2447	1.9057	0.9973	0.7328	0.819
d_{19}	0.7074	2.0859	1.1962	1.3753	2.1257	1.2241	2.2533	1.8633	1.2587	0.8414	1.1186	1.5131	2.0393	1.2856	0.7939	0.8601	0.45	1.6645	1.5023	1.5358
d_{20}	1.7443	0.885	2.0351	0.8697	0.8509	1.6918	0.8763	0.9174	1.969	1.8306	1.9693	0.85	0.7676	0.6918	1.2557	1.2515	1.7036	0.8171	0.8468	0.8649
d_{21}	1.8185	1.1031	2.0052	1.1756	0.7307	1.7995	0.9036	0.6344	2.0666	2.0027	2.0088	0.8041	0.6107	0.8805	1.5431	1.3361	1.9116	1.1412	0.7639	0.961
d_{22}	1.9574	0.9255	1.7039	1.1971	1.1751	1.5944	1.2097	1.4736	1.6067	1.5854	1.7154	1.2933	1.1488	1.105	1.8622	1.7412	2.0216	0.5219	1.3402	0.7414
d_{23}	1.8568	0.7363	1.8082	1.2147	0.6156	1.401	0.6694	0.9875	1.8234	1.8198	1.8362	1.0494	0.6069	1.0116	1.6667	1.4042	2.0084	0.8769	0.9878	0.8119
d_{24}	1.6282	0.8407	1.6651	0.8992	0.9741	1.4972	1.0802	1.138	1.7211	1.4649	1.6866	0.9559	0.9306	0.7678	1.5029	1.416	1.7272	0.5236	1.0105	0.5595
d_{25}	1.3359	1.1445	1.9062	0.6743	1.1183	1.8032	1.3148	0.8875	2.0079	1.5999	1.868	0.6086	1.0329	0.3919	1.3992	1.392	1.4505	0.8565	0.6427	0.7017
d_{26}	0.8122	1.975	1.0348	1.3499	2.0864	1.1603	2.0999	1.9844	1.0309	0.6843	0.8951	1.6892	2.0216	1.4005	0.7956	0.924	0.6031	1.4383	1.6158	1.3178
d_{27}	1.5739	1.3072	1.8111	0.8839	1.056	1.896	0.9935	0.7108	1.7815	1.8277	1.7134	0.7657	0.9799	0.6719	1.3182	1.4605	1.4873	1.0563	0.4852	0.8406
d_{28}	1.5586	1.1975	2.011	1.0052	0.9534	1.7683	1.1907	0.61	2.0967	1.8168	1.9963	0.5811	0.8272	0.6857	1.4181	1.2667	1.7263	1.0918	0.7165	0.8659
d_{29}	1.5907	1.1043	1.744	0.8225	1.1337	1.7462	1.0153	0.9852	1.7049	1.6495	1.6626	0.9578	1.0838	0.7115	1.2214	1.4387	1.4121	0.8012	0.7616	0.7871
d_{30}	1.5263	1.183	1.4796	0.8761	1.148	1.6767	1.2694	1.1526	1.5908	1.2705	1.4303	0.9681	1.0961	0.8267	1.6365	1.5151	1.6019	0.6525	0.9112	0.3224
d_{31}	1.631	1.5417	1.9223	0.8616	1.3312	2.1135	1.2552	0.8491	1.903	1.9033	1.7924	0.749	1.2463	0.683	1.4996	1.6809	1.3058	1.2014	0.4	0.9783
d_{32}	1.7091	0.8696	1.9546	1.0093	0.8499	1.4998	0.9029	1.113	1.8869	1.7428	1.9094	1.0149	0.7865	0.8458	1.2786	1.2277	1.7139	0.8572	1.0552	0.9355
d_{33}	2.0178	1.1003	2.1701	1.3144	0.8966	1.8013	0.8779	1.053	2.083	2.1133	2.1136	1.1823	0.7245	1.0995	1.4489	1.4987	1.8412	1.2285	1.1043	1.3086
d_{34}	1.2912	1.5601	0.9931	2.0385	1.6291	0.7585	1.6253	1.991	0.9332	1.0447	1.0201	1.9961	1.6238	1.8979	1.1933	0.9746	1.4151	1.4455	1.911	1.4985
d_{35}	0.9406	2.1711	1.0449	1.3949	2.0143	1.3852	1.9025	1.6845	1.0291	1.0079	0.8894	1.6324	1.9568	1.4567	0.9239	1.0808	0.5583	1.6792	1.2648	1.399
d_{36}	1.1016	2.1863	1.39	1.7441	1.8617	1.4462	1.8741	1.2849	1.4133	1.4097	1.2846	1.2661	1.7496	1.4866	0.89	0.9532	0.9435	2.0021	1.1486	1.7227
d_{37}	1.2715	1.6771	0.8886	1.7909	1.7565	0.9589	1.5382	1.8632	0.7599	1.0262	0.8451	1.9655	1.7442	1.7308	0.9079	1.0315	1.0609	1.3203	1.6587	1.4492
d_{38}	1.5066	1.39	1.7587	0.7725	1.4901	1.9096	1.6146	1.3291	1.8571	1.4035	1.6527	0.9982	1.4205	0.7749	1.6178	1.6413	1.2728	0.874	0.9868	0.7261
d_{39}	1.7152	0.9037	1.5608	1.2569	0.8662	1.302	0.8837	1.1213	1.572	1.5983	1.5899	1.1668	0.8544	1.0942	1.5689	1.3658	1.8292	0.8523	1.0561	0.8088
d_{40}	1.6129	1.1017	1.5645	1.1743	0.9183	1.5073	0.9109	0.9188	1.6023	1.6466	1.5544	1.049	0.8795	0.9808	1.3286	1.2578	1.6371	0.9726	0.8401	0.8157
d_{41}	1.9492	0.9292	1.5999	1.5186	1.0031	1.3393	0.8698	1.2853	1.5553	1.7616	1.6278	1.4354	0.9985	1.3559	1.7078	1.5743	1.9666	0.9841	1.2418	1.0105
d_{42}	1.8856	0.9363	1.6617	1.2383	1.2226	1.4146	1	1.4491	1.5005	1.6318	1.6194	1.4131	1.1947	1.1731	1.4682	1.5904	1.7207	0.7415	1.2774	0.9894
d_{43}	1.7428	1.0657	1.5911	1.1342	1.0339	1.5218	0.8706	1.1024	1.5098	1.6814	1.5378	1.2275	1.0091	1.0249	1.3089	1.4136	1.5742	0.8911	0.9446	0.9045
d_{44}	1.8874	1.2498	1.9639	1.346	0.9028	1.7521	0.8857	0.9245	1.891	2.0399	1.9022	1.1572	0.7606	1.1289	1.3667	1.3685	1.7536	1.305	0.969	1.2424
d_{45}	0.7336	1.9494	1.2643	1.3394	1.833	1.2339	2.0227	1.5482	1.3711	0.9934	1.2061	1.2231	1.7385	1.1339	0.9147	0.8339	0.7534	1.611	1.1927	1.3365
d_{46}	1.8955	1.0454	1.602	1.2704	1.3332	1.488	1.121	1.5392	1.4134	1.5783	1.5538	1.4803	1.3067	1.2213	1.51	1.6434	1.7182	0.7069	1.3427	0.9869
d_{47}	1.398	1.247	1.4351	0.9757	1.1102	1.5731	1.2137	1.0664	1.534	1.3591	1.4145	1.0019	1.0703	0.8693	1.4628	1.3461	1.4911	0.8505	0.8617	0.584
d_{48}	1.4525	1.5281	1.3239	2.0628	1.282	0.9692	1.148	1.6828	1.2552	1.4429	1.2809	1.8907	1.257	1.8802	1.0545	0.9433	1.4321	1.7921	1.6787	1.7283
d_{49}	0.9708	1.9057	1.0504	1.3479	1.8451	1.3181	1.717	1.605	1.0254	1.014	0.9308	1.613	1.7952	1.332	0.8035	1.0367	0.721	1.4703	1.2677	1.3023
d_{50}	1.3743	1.276	1.9168	0.9599	1.1818	1.7046	1.4048	0.8422	2.0108	1.6266	1.8917	0.6317	1.068	0.6327	1.255	1.2181	1.4652	1.0991	0.7825	0.9658
d_{51}	1.6619	1.225	1.2753	1.2316	1.1829	1.5137	1.0879	1.2045	1.2796	1.4044	1.2595	1.2805	1.1646	1.1571	1.5073	1.4889	1.5956	0.7633	1.0051	0.7161
d_{52}	1.8204	1.1368	1.5315	1.2066	1.408	1.5429	1.2892	1.6289	1.3125	1.4306	1.4695	1.4604	1.3788	1.1881	1.5676	1.6417	1.6922	0.6001	1.3875	0.9028
d_{53}	1.663	1.419	2.0847	1.1355	1.3046	1.8769	1.3162	0.8078	2.0448	1.9036	1.9949	0.861	1.1609	0.8759	1.2122	1.3712	1.4919	1.3036	0.8956	1.2714
d_{54}	1.2314	2.2243	1.1141	1.5597	2.1032	1.5783	1.9987	1.8191	1.1031	1.0634	0.9112	1.7618	2.0581	1.6224	1.2333	1.3491	0.8229	1.6401	1.3934	1.3589
d_{55}	1.5722	1.2955	1.1955	1.1913	1.2373	1.5441	1.2232	1.2286	1.2874	1.2639	1.2049	1.2481	1.2136	1.1152	1.5399	1.471	1.5474	0.749	1.0298	0.6174
d_{56}	1.5281	1.5803	2.1028	1.1848	1.4046	1.9075	1.6439	0.9635	2.2027	1.8068	2.0637	0.6653	1.2671	0.8772	1.5076	1.4072	1.5577	1.4061	0.8898	1.205
d_{57}	1.0141	1.6802	1.4625	1.6085	1.4652	1.1238	1.7056	1.2639	1.5383	1.2392	1.456	1.2169	1.3348	1.294	0.814	0.6188	1.2008	1.6314	1.3864	1.4695
d_{58}	1.6971	1.0762	1.8687	1.2378	0.9774	1.4936	0.9564	1.1532	1.7898	1.7611	1.8126	1.1917	0.8911	1.0485	1.1341	1.178	1.5837	1.1171	1.1329	1.1909
d_{59}	1.6462	1.6228	1.5502	1.0009	1.6952	1.9549	1.5462	1.4577	1.5227	1.397	1.3526	1.3469	1.6544	1.0678	1.5113	1.7481	1.1776	1.0069	1.0544	0.9204

Table 6.1: Anchor Point Analysis for painting: Nevermore, Paul Gauguin (1897)

	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}	t_{37}	t_{38}	t_{39}
d_0	1.7178	2.075	0.7543	1.6917	1.4465	1.0868	0.4204	2.1348	0.6054	0.7629	0.7165	0.8892	1.7589	1.0084	1.407	1.747	0.4276	0.9854	0.7099	0.6668
d_1	1.6016	1.7543	0.9356	1.5285	1.9109	0.8182	0.7215	1.6321	0.463	0.35	0.7869	0.8288	1.6548	0.6493	1.6005	1.2741	0.6709	0.7218	0.7276	0.5523
d_2	2.0384	1.9902	0.3991	1.8024	1.5501	0.8309	0.1	1.9341	0.5193	0.604	0.3061	0.6407	1.8271	0.8584	1.5527	1.7774	0.6593	1.0573	0.6981	0.4158
d_3	1.6123	1.6181	1.108	1.7533	1.7548	0.849	0.9355	1.701	0.4821	0.4853	0.9606	0.7637	1.5546	0.5624	1.5705	1.3261	0.5329	0.6398	0.5054	0.7241
d_4	1.8753	2.0197	0.5726	1.6326	1.4851	0.9111	0.1598	1.9611	0.5081	0.6327	0.467	0.7354	1.7903	0.9148	1.4579	1.6782	0.599	0.9984	0.7661	0.4834
d_5	0.831	0.4794	1.7738	0.7808	0.7992	1.499	1.7707	0.41	1.666	1.3928	1.6173	1.4878	0.15	1.4923	0.48	0.4173	1.8399	1.4488	1.4413	1.4602
d_6	1.3207	2.0806	1.142	1.3324	1.782	1.1712	0.7716	2.0177	0.6883	0.7686	0.984	1.089	1.8096	1.0456	1.5665	1.575	0.5742	0.8902	0.8837	0.8267
d_7	0.34	0.9836	2.0564	0.14	0.7024	2.0937	1.675	0.8911	1.6147	1.6766	1.8907	2.0135	0.753	1.9619	0.5074	0.4494	1.4859	1.8984	1.8318	1.621
d_8	1.9807	2.1274	0.6969	1.9553	1.5508	1.1973	0.4625	2.2384	0.8424	0.9835	0.6247	0.8514	1.8118	1.1139	1.5503	1.9293	0.713	1.102	0.7551	0.7698
d_9	1.116	0.8919	1.5266	1.0033	0.47	1.9454	1.4149	1.0499	1.639	1.8641	1.4727	1.5999	0.8774	1.8547	0.6632	0.9848	1.5488	2.1626	1.5528	1.5193
d_{10}	2.1127	1.7547	0.6102	2.0035	1.9382	0.516	0.786	1.6539	0.8198	0.5924	0.4996	0.4046	1.4108	0.651	1.6295	1.601	1.0113	0.6227	0.5801	0.5712
d_{11}	0.7531	0.5797	1.9417	0.6994	0.897	1.6251	1.7934	0.4247	1.4305	1.3515	1.7951	1.6886	0.6564	1.4589	0.7223	0.31	1.6282	1.55	1.5835	1.4857
d_{12}	0.9476	0.5483	1.7834	0.9499	0.9687	1.3564	1.7755	0.4314	1.6283	1.3133	1.63	1.4866	0.2608	1.3028	0.5978	0.4591	1.773	1.2093	1.4031	1.465
d_{13}	1.5145	1.8863	0.8673	1.3753	1.3596	1.0491	0.4752	1.8182	0.5816	0.6771	0.7085	0.8997	1.5513	0.9672	1.2048	1.4171	0.3765	0.8755	0.6857	0.5265
d_{14}	1.968	1.6158	0.7638	1.9642	1.8543	0.4816	0.8731	1.5522	0.8425	0.5899	0.6352	0.4754	1.2433	0.625	1.522	1.5258	0.9699	0.4755	0.5291	0.6321
d_{15}	1.3131	0.8357	1.7482	1.3865	0.9969	1.5103	1.9731	0.923	1.8343	1.7814	1.6849	1.3914	0.6235	1.5932	0.8594	0.9558	1.8612	1.6208	1.457	1.7224
d_{16}	1.5426	1.412	1.1747	1.6755	1.6552	0.8508	1.0043	1.4905	0.5833	0.516	0.997	0.7756	1.3367	0.5625	1.4308	1.1654	0.617	0.5487	0.4966	0.7143
d_{17}	1.0123	0.6061	1.5173	1.0551	0.5524	1.5292	1.5605	0.7549	1.3851	1.4823	1.3746	1.2108	0.494	1.4356	0.5295	0.7405	1.3928	1.6481	1.0591	1.2389
d_{18}	1.9758	1.7178	0.3707	1.7711	1.6014	0.75	0.4406	1.6569	0.6657	0.594	0.2	0.5112	1.5005	0.7871	1.4342	1.5711	0.8212	0.9084	0.5808	0.3595
d_{19}	1.07	0.6592	1.8283	1.1725	1.0773	1.2539	1.877	0.5951	1.7322	1.4454	1.6897	1.4687	0.3861	1.3392	0.7231	0.6742	1.805	1.1728	1.414	1.5752
d_{20}	1.7622	1.6204	0.6933	1.7805	1.3728	0.8875	0.6201	1.775	0.6387	0.7409	0.5197	0.5542	1.3582	0.7863	1.249	1.546	0.604	0.8355	0.3885	0.4847
d_{21}	2.0612	1.7444	0.28	1.8526	1.6091	0.7925	0.5217	1.6806	0.7506	0.7208	0.2208	0.5603	1.5795	0.8603	1.5185	1.6617	0.9012	1.0238	0.6687	0.4496
d_{22}	1.1184	1.8322	1.2422	1.19	1.6114	1.2225	0.8995	1.7615	0.7606	0.779	1.0678	1.1603	1.5636	1.0581	1.3598	1.313	0.5702	0.9058	0.8478	0.769
d_{23}	1.7021	1.7268	0.6769	1.4711	1.2594	0.9506	0.2973	1.668	0.5677	0.6273	0.5237	0.76	1.5576	0.899	1.2297	1.4548	0.5639	1.0194	0.6816	0.3822
d_{24}	1.4525	1.5531	0.933	1.4352	1.4439	0.9205	0.7048	1.534	0.45	0.5094	0.7542	0.816	1.2291	0.8018	1.0826	1.1613	0.5371	0.6796	0.4991	0.45
d_{25}	1.807	1.4816	0.8305	1.7986	1.7049	0.5933	0.8863	1.4322	0.8364	0.574	0.6828	0.534	1.0866	0.658	1.3655	1.3632	0.9396	0.5453	0.5094	0.5972
d_{26}	0.9101	0.4787	1.8902	1.0365	0.9657	1.4637	1.8172	0.4861	1.5047	1.2821	1.7277	1.4792	0.4028	1.1925	0.6819	0.5273	1.5756	1.1175	1.29	1.509
d_{27}	1.9814	1.3414	0.6744	1.8619	1.566	0.5251	0.7961	1.4041	0.5712	0.5948	0.553	0.23	1.3418	0.5	1.5286	1.5159	0.816	0.8399	0.3975	0.487
d_{28}	1.9703	1.6813	0.5539	1.8036	1.7191	0.7637	0.7567	1.6027	0.8962	0.7324	0.4525	0.5469	1.3507	0.8594	1.4462	1.4934	1.044	0.8655	0.693	0.5748
d_{29}	1.6875	1.2925	0.88	1.7426	1.3382	0.7452	0.8551	1.4479	0.5348	0.5993	0.7327	0.5002	1.1733	0.5852	1.2587	1.3777	0.5742	0.7768	0.19	0.4769
d_{30}	1.5743	1.31	1.0196	1.4706	1.7147	0.7464	0.8777	1.2028	0.6271	0.3564	0.8449	0.7873	1.1991	0.5836	1.3492	0.9923	0.8742	0.5622	0.6783	0.5815
d_{31}	2.1	1.3388	0.8927	2.0465	1.7465	0.42	1.0759	1.3876	0.8074	0.7518	0.7994	0.4186	1.398	0.5043	1.698	1.6282	1.0264	0.7954	0.5997	0.7529
d_{32}	1.607	1.6103	0.8194	1.609	1.2182	1.0935	0.6135	1.7404	0.7436	0.8408	0.6549	0.7627	1.3132	0.9557	1.0978	1.446	0.6076	0.9793	0.5394	0.5411
d_{33}	1.9491	1.783	0.7131	1.9352	1.3242	1.1797	0.7337	1.9364	0.9305	1.1338	0.6883	0.8281	1.6453	1.1075	1.4632	1.8205	0.7698	1.2884	0.7072	0.8043
d_{34}	0.689	1.0929	1.714	0.4352	0.7927	1.8145	1.3189	1.0075	1.3232	1.4079	1.5524	1.7046	0.965	1.6993	0.6669	0.6903	1.1832	1.7578	1.5702	1.3012
d_{35}	1.2857	0.4983	1.678	1.2344	1.0806	1.0894	1.7295	0.4631	1.3324	1.2255	1.5529	1.1913	0.6705	0.9069	1.0047	0.7692	1.5826	1.2093	1.2332	1.3582
d_{36}	1.6125	0.9464	1.3518	1.4433	1.1392	1.2561	1.6345	0.9425	1.5286	1.4961	1.301	1.1154	0.9313	1.3654	1.1282	1.1007	1.7232	1.6477	1.3404	1.3577
d_{37}	0.7896	0.7409	1.6846	0.7501	0.6432	1.601	1.4334	0.8709	1.0332	1.3309	1.5407	1.4024	0.8422	1.3786	0.6207	0.6808	1.0293	1.6407	1.2087	1.2315
d_{38}	1.6862	1.3309	1.263	1.7739	1.815	0.7484	1.2275	1.1485	1.0188	0.6961	1.1068	0.934	1.1432	0.6186	1.4469	1.2518	1.1038	0.26	0.7928	0.9233
d_{39}	1.4912	1.5199	0.8533	1.2811	1.2448	0.9841	0.5441	1.4556	0.5313	0.6397	0.689	0.8416	1.3991	0.9083	1.0562	1.2356	0.567	1.0217	0.7308	0.4558
d_{40}	1.6906	1.3726	0.7455	1.4976	1.285	0.7947	0.6334	1.3551	0.5178	0.6122	0.5954	0.619	1.3178	0.7428	1.1651	1.2903	0.6729	1.005	0.5095	0.28
d_{41}	1.4548	1.6492	1.044	1.1995	1.2409	1.1681	0.6902	1.5941	0.683	0.8781	0.9146	1.0253	1.6294	1.0659	1.2927	1.3969	0.5087	1.2748	0.8996	0.6718
d_{42}	1.2504	1.536	1.2386	1.243	1.1597	1.2371	0.9302	1.6679	0.6589	0.965	1.0991	1.0088	1.4438	1.0125	1.1506	1.3344	0.35	1.1031	0.7075	0.8153
d_{43}	1.5581	1.2888	0.9166	1.4921	1.142	0.9299	0.7277	1.4408	0.4591	0.7184	0.7703	0.6668	1.3321	0.7521	1.0662	1.3292	0.5221	1.0525	0.4894	0.4712
d_{44}	1.9434	1.5678	0.591	1.82	1.3206	1.0477	0.7192	1.7002	0.8456	1.0204	0.5545	0.6935	1.5682	0.98	1.4373	1.6897	0.8525	1.3241	0.7214	0.6947
d_{45}	1.2832	0.8493	1.5167	1.1961	1.2207	1.0676	1.5938	0.7186	1.5182	1.228	1.3813	1.2163	0.5554	1.2233	0.8761	0.7272	1.7061	1.1549	1.2973	1.2949
d_{46}	1.133	1.5288	1.3459	1.1609	1.2407	1.2836	1.0433	1.6473	0.7179	0.9961	1.2024	1.0859	1.4481	1.0405	1.2035	1.2891	0.4696	1.1108	0.7762	0.8997
d_{47}	1.5793	1.1824	0.9244	1.4426	1.511	0.7378	0.8167	1.1231	0.6133	0.3552	0.751	0.7057	1.0642	0.6291	1.2251	1.0369	0.8812	0.7462	0.5977	0.4982
d_{48}	1.2755	1.1704	1.3752	1.0317	0.6503	1.6956	1.0367	1.1971	1.3168	1.5439	1.2882	1.4494	1.1608	1.6361	0.8513	1.0773	1.2476	1.9261	1.4014	1.2174
d_{49}	1.1725	0.47	1.5532	1.1919	0.9252	1.1734	1.5544	0.6218	1.1623	1.0769	1.4174	1.0866	0.5334	0.9781	0.8455	0.7604	1.3286	1.2622	0.9817	1.1883
d_{50}	1.7873	1.522	0.8125	1.7523	1.588	0.8256	0.9693	1.4722	1.037	0.8349	0.6934	0.6326	1.1163	0.9346	1.2875	1.3765	1.0915	0.8126	0.7003	0.7344
d_{51}	1.4332	1.264	1.0601	1.2993	1.3447	0.8885	0.8684	1.2363	0.4557	0.6057	0.9114	0.8285	1.3283	0.7486	1.2245	1.0375	0.6693	0.9754	0.6907	0.5959
d_{52}	1.0458	1.5008	1.4338	1.1985	1.3712	1.3015	1.1146	1.6012	0.7899	0.9379	1.2599	1.1503	1.3689	1.0451	1.2105	1.1641	0.6445	1.0036	0.847	0.9718
d_{53}	1.9354	1.5535	0.8125	1.9636	1.4876	1.0034	1.12	1.6817	1.0875	1.1406	0.7903	0.7007	1.3441	1.0173	1.4323	1.6438	1.0571	1.1158	0.7272	0.9307
d_{54}	1.3906	0.7311	1.8054	1.3478	1.3244	1.2001	1.8059	0.5817	1.3661	1.2587	1.6751	1.356	0.9597	1.0287	1.2388	0.871	1.6252	1.0776	1.362	1.4433
d_{55}	1.4216	1.2143	1.0793	1.3123	1.4413	0.8847	0.9269	1.1152	0.5761	0.5395	0.9304	0.8742	1.2385	0.7363	1.2412	0.8964	0.8085	0.8801	0.7478	0.6184
d_{56}	2.0389	1.7074	0.9644	1.9605	1.831	0.9311	1.2082	1.5912	1.304	1.0777	0.902	0.8897	1.3357	1.1141	1.5417	1.5626	1.4085	0.9981	1.0147	1.0068
d_{57}	1.3485	1.1711	1.0673	1.2159	1.0619	1.4209	1.2691	1.099	1.4666	1.3437	0.9625	1.1914	0.7882	1.5001	0.8116	0.9272	1.5338	1.456	1.2448	1.1306
d_{58}	1.6201	1.499	0.8757	1.6165	1.0807	1.1835	0.7736	1.6539	0.8797	1.0831	0.759	0.8414	1.3115	1.074	1.1164	1.4656	0.7343	1.2045	0.6605	0.7392
d_{59}	1.6318	1.1099	1.4263	1.7768	1.5724	0.878	1.399	1.067	0.9296	0.8445	1.2885	0.9409	1.2427	0.6213	1.4925	1.2628	0.9976	0.621	0.7612	1.0343

Table 6.2: Anchor Point Analysis for painting: Nevermore, Paul Gauguin (1897)

	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}	t_{46}	t_{47}	t_{48}	t_{49}	t_{50}	t_{51}	t_{52}	t_{53}	t_{54}	t_{55}	t_{56}	t_{57}	t_{58}	t_{59}
d_0	1.1852	1.5774	0.7612	0.6503	0.72	0.7106	1.7134	2.1671	0.8021	0.7649	1.262	0.9403	1.2122	0.5917	0.7508	0.7345	1.8611	0.6003	1.3778	1.4102
d_1	1.0073	1.484	0.412	0.301	0.9878	0.5823	1.8374	2.0186	0.3987	1.1371	1.1462	1.0043	0.9111	0.7122	0.8478	0.7567	1.5305	0.7779	1.4054	1.9144
d_2	0.9403	1.6428	0.5932	0.7539	0.3684	0.5109	1.5144	1.8175	0.6902	0.6581	1.0201	0.8774	0.9143	0.878	0.3478	0.4251	2.0369	0.7654	1.3713	1.2555
d_3	0.9236	1.4453	0.569	0.4896	1.1905	0.6565	1.8668	1.9973	0.5039	0.9244	1.0437	0.9257	0.9229	0.571	1.0357	0.916	1.3022	0.6301	1.5688	1.7413
d_4	1.0338	1.5935	0.6	0.6104	0.4158	0.5279	1.5697	1.9399	0.6692	0.7351	1.1238	0.9131	0.9965	0.7596	0.4786	0.4982	1.9628	0.7142	1.2834	1.3608
d_5	1.5615	0.7658	1.3881	1.5748	1.8169	1.4847	0.5666	0.7137	1.5234	1.6758	1.5287	1.3161	1.2849	1.7263	1.4858	1.4244	0.7302	1.5908	0.9859	1.1201
d_6	1.3147	1.5624	0.7067	0.3601	0.9887	0.7815	1.9164	2.3183	0.6811	1.1001	1.4288	1.116	1.2458	0.5005	0.9837	0.9202	1.7458	0.7013	1.324	1.7611
d_7	2.1705	0.5106	1.4975	1.1804	1.7704	1.5427	0.9384	1.2821	1.4028	1.9028	2.1294	1.8798	1.8936	1.2255	1.5861	1.5181	1.0117	1.363	0.4789	1.0577
d_8	1.0779	1.7633	0.9081	0.8967	0.26	0.8042	1.5059	1.9701	1.0026	0.43	1.122	0.8225	1.1652	0.8235	0.6266	0.7329	1.9583	0.7827	1.6399	1.0066
d_9	1.7945	0.6098	1.6924	1.8851	1.2578	1.4697	0.6636	0.8002	1.6268	1.2227	1.6682	1.6566	1.6744	1.5602	1.2054	1.2867	1.2446	1.3778	0.767	0.3324
d_{10}	0.4165	1.8695	0.6101	0.8003	0.785	0.6444	1.2309	1.2601	0.7916	0.6611	0.4408	0.2	0.4251	1.066	0.6501	0.651	1.6034	0.9832	1.7654	1.715
d_{11}	1.7404	0.4977	1.2392	1.3197	1.9508	1.3735	0.9907	0.9259	1.1739	1.9702	1.7281	1.779	1.466	1.4887	1.5531	1.4251	0.6365	1.4059	0.6613	1.1629
d_{12}	1.4655	0.7949	1.3341	1.5055	1.8299	1.4713	0.6158	0.6984	1.465	1.6467	1.4538	1.2548	1.1934	1.6438	1.498	1.4324	0.5345	1.5205	0.9575	1.0994
d_{13}	1.0916	1.3691	0.4879	0.3633	0.5931	0.4432	1.4736	1.9292	0.4898	0.6896	1.1061	0.7695	0.922	0.4165	0.5222	0.4844	1.6618	0.4007	1.0728	1.354
d_{14}	0.4411	1.7672	0.5778	0.7647	0.8757	0.6448	1.2818	1.3576	0.7577	0.6703	0.443	0.2417	0.3247	0.963	0.6929	0.6713	1.4123	0.867	1.7426	1.6744
d_{15}	1.238	0.9732	1.7645	1.9534	1.9158	1.684	0.5897	0.2681	1.7923	1.4148	1.0625	1.1037	1.2666	1.771	1.6391	1.6161	0.8333	1.6186	1.2194	0.9724
d_{16}	0.849	1.3666	0.48	0.4959	1.1224	0.5914	1.6783	1.8045	0.4311	0.7911	0.891	0.7188	0.7371	0.5132	0.8837	0.7815	1.0807	0.546	1.5002	1.6278
d_{17}	1.4144	0.5152	1.3632	1.6477	1.6367	1.2131	0.5166	0.5377	1.365	1.2562	1.3168	1.2518	1.2656	1.3746	1.2126	1.1404	0.7639	1.1564	0.7342	0.5808
d_{18}	0.707	1.5743	0.4908	0.7322	0.4391	0.4159	1.1029	1.4197	0.5927	0.4239	0.6628	0.4709	0.5569	0.892	0.21	0.2692	1.7495	0.7226	1.4004	1.339
d_{19}	1.3246	0.9147	1.4651	1.6357	1.9114	1.5925	0.7213	0.709	1.5973	1.6079	1.3039	1.1789	1.1556	1.6786	1.5864	1.5348	0.5228	1.5507	1.0951	1.1114
d_{20}	0.7606	1.4249	0.6204	0.727	0.6442	0.4556	1.2088	1.5379	0.6438	0.2996	0.6905	0.4508	0.6561	0.5934	0.3851	0.3743	1.4867	0.4746	1.4454	1.1766
d_{21}	0.7321	1.6064	0.5814	0.8647	0.4684	0.5044	1.0666	1.3604	0.6785	0.4006	0.637	0.5161	0.5589	0.9553	0.218	0.3185	1.8325	0.7613	1.4184	1.2866
d_{22}	1.3112	1.3416	0.6401	0.29	1.0154	0.7128	1.6966	2.0806	0.5263	0.9859	1.3228	0.923	1.0684	0.3336	0.8283	0.7363	1.486	0.5383	1.1786	1.6079
d_{23}	0.9647	1.3283	0.4598	0.5658	0.4578	0.3617	1.3756	1.6657	0.4673	0.6376	0.9213	0.735	0.7215	0.607	0.2396	0.2184	1.7327	0.4527	1.072	1.2101
d_{24}	0.9822	1.2388	0.3465	0.4114	0.8167	0.3813	1.363	1.73	0.3529	0.6854	0.9648	0.5631	0.7357	0.4255	0.5405	0.4259	1.3309	0.3823	1.2351	1.4133
d_{25}	0.6179	1.6083	0.5707	0.7658	0.9289	0.6435	1.1963	1.3649	0.7091	0.675	0.5876	0.2764	0.3726	0.8714	0.597	0.579	1.2983	0.7637	1.5818	1.5525
d_{26}	1.4702	0.7265	1.3042	1.4311	1.8949	1.4516	0.8342	0.8673	1.3938	1.5683	1.44	1.3164	1.2975	1.4241	1.565	1.4823	0.28	1.3268	0.9256	0.9827
d_{27}	0.4256	1.4008	0.5174	0.8648	0.8512	0.4204	1.2997	1.1511	0.5473	0.6318	0.3684	0.4785	0.262	0.8119	0.472	0.4081	1.4752	0.5927	1.3942	1.3973
d_{28}	0.6811	1.6801	0.6774	0.8588	0.6939	0.655	0.9554	1.2144	0.7881	0.5255	0.5892	0.2427	0.5075	0.9804	0.4249	0.4564	1.5901	0.8431	1.5423	1.4534
d_{29}	0.6707	1.2309	0.4917	0.7617	0.966	0.3944	1.3472	1.4047	0.4973	0.5725	0.6238	0.535	0.5043	0.5174	0.5497	0.4228	1.287	0.3529	1.3272	1.2329
d_{30}	0.8517	1.3699	0.335	0.4764	0.977	0.5338	1.4535	1.5982	0.4098	0.9544	0.8591	0.6831	0.5764	0.7203	0.646	0.5518	1.1366	0.662	1.3222	1.6713
d_{31}	0.22	1.5427	0.7217	1.0764	1.1144	0.6782	1.4536	1.0515	0.7543	0.8175	0.2	0.5099	0.18	0.9945	0.7264	0.6675	1.3994	0.769	1.6004	1.5724
d_{32}	0.9755	1.322	0.7161	0.7777	0.6737	0.5691	1.169	1.566	0.7303	0.4063	0.9029	0.5646	0.8393	0.549	0.3694	0.3417	1.434	0.4414	1.3105	1.043
d_{33}	1.0111	1.5083	0.9733	1.1148	0.5277	0.7539	1.2648	1.5356	0.9205	0.27	0.8888	0.735	0.9036	0.7612	0.4789	0.5683	1.7774	0.5851	1.437	0.8565
d_{34}	1.8778	0.5421	1.2218	1.0727	1.4453	1.2299	0.9711	1.2126	1.1397	1.6601	1.8289	1.6713	1.6049	1.0236	1.2411	1.1822	0.9963	1.0804	0.19	0.7821
d_{35}	1.1162	0.7793	1.2222	1.5438	1.8315	1.254	0.958	0.6402	1.237	1.6055	1.1175	1.373	0.9423	1.5074	1.4058	1.305	0.5114	1.3338	0.8408	0.9869
d_{36}	1.0631	1.0454	1.4409	1.7921	1.6052	1.3356	0.6311	0.24	1.4596	1.3545	0.9229	1.0448	0.9935	1.7061	1.2566	1.2443	1.0264	1.4952	1.0227	0.9241
d_{37}	1.5587	0.21	1.1125	1.1326	1.5946	1.038	1.0325	0.9969	0.9293	1.4647	1.4923	1.5623	1.3778	0.9301	1.2456	1.1268	0.786	0.8065	0.3442	0.6511
d_{38}	0.8424	1.5834	0.7183	0.8296	1.2954	0.8962	1.4868	1.5794	0.8219	1.0548	0.8305	0.6572	0.6336	0.9502	0.9671	0.8897	0.8801	0.8442	1.6276	1.7353
d_{39}	1.0278	1.0781	0.4374	0.5356	0.704	0.3782	1.3553	1.5715	0.4286	0.8623	0.9713	0.8581	0.7681	0.5656	0.4059	0.3313	1.5281	0.3273	0.8961	1.2413
d_{40}	0.8174	1.1521	0.4512	0.7235	0.7605	0.27	1.2409	1.3336	0.4199	0.7162	0.7725	0.7179	0.5616	0.6563	0.3409	0.21	1.4915	0.4547	1.0199	1.1925
d_{41}	1.215	1.1102	0.6644	0.6304	0.8186	0.558	1.5545	1.7141	0.5232	0.982	1.1697	1.1027	0.947	0.4487	0.5788	0.5059	1.6945	0.4286	0.6877	1.2132
d_{42}	1.183	0.9729	0.7515	0.5713	1.0327	0.6332	1.5471	1.7583	0.5654	0.8129	1.1241	0.9789	1.0073	0.17	0.7875	0.6796	1.3758	0.22	0.892	1.155
d_{43}	0.854	0.9657	0.5219	0.7408	0.8762	0.303	1.3989	1.4157	0.4488	0.7	0.7819	0.8154	0.6836	0.5008	0.4879	0.3591	1.3846	0.2779	1.0124	1.0982
d_{44}	0.8765	1.3625	0.8762	1.1262	0.6111	0.6638	1.1528	1.2912	0.8353	0.3249	0.7553	0.759	0.7725	0.8594	0.3967	0.4758	1.7038	0.6552	1.2921	0.9432
d_{45}	1.1103	1.0638	1.237	1.4212	1.6101	1.3324	0.6505	0.6503	1.3711	1.4522	1.0982	0.9926	0.8388	1.5935	1.2898	1.2482	0.6738	1.4577	0.993	1.0692
d_{46}	1.2437	0.9882	0.7907	0.5193	1.1481	0.7155	1.6	1.7991	0.5856	0.9144	1.1931	1.0462	1.0612	0.2106	0.8916	0.781	1.323	0.3304	0.9492	1.2375
d_{47}	0.8163	1.2139	0.32	0.646	0.9471	0.4715	1.2832	1.3595	0.4526	0.961	0.7949	0.7059	0.5367	0.8139	0.5595	0.47	1.2022	0.6328	1.1547	1.4493
d_{48}	1.6546	0.7627	1.3468	1.4664	1.0452	1.1277	0.8692	1.0137	1.2473	1.219	1.5659	1.5506	1.4337	1.2518	0.9598	0.988	1.2893	1.0996	0.4847	0.32
d_{49}	1.1494	0.6416	1.0588	1.4057	1.6689	1.0763	0.9377	0.7437	1.0861	1.3819	1.1016	1.2329	0.9654	1.2629	1.2441	1.1356	0.6073	1.0755	0.769	0.8458
d_{50}	0.7509	1.5687	0.7972	0.977	0.9367	0.8078	0.9314	1.1317	0.9204	0.6631	0.6437	0.229	0.5675	1.0077	0.6413	0.6447	1.3413	0.8546	1.5071	1.3631
d_{51}	0.962	0.9449	0.3992	0.5274	1.0274	0.4105	1.4657	1.4652	0.2	1.0146	0.9347	0.9737	0.6825	0.5665	0.6418	0.5148	1.2943	0.4306	0.9128	1.3379
d_{52}	1.278	1.0614	0.7771	0.4207	1.2246	0.8082	1.595	1.8309	0.6238	1.0063	1.2313	1.04	1.0928	0.3981	0.983	0.8937	1.1707	0.4618	1.099	1.3706
d_{53}	0.7783	1.5364	1.0591	1.2459	1.0083	0.9106	0.968	1.065	1.0643	0.4988	0.6304	0.4348	0.7279	0.9867	0.7583	0.7782	1.4662	0.8311	1.5723	1.1794
d_{54}	1.2557	0.9603	1.2406	1.4794	1.9318	1.3101	1.2423	0.9231	1.2189	1.7587	1.2631	1.5265	1.072	1.5175	1.5057	1.393	0.4384	1.3804	0.9865	1.2697
d_{55}	0.9783	1.0468	0.3764	0.5574	1.0806	0.5162	1.4353	1.4635	0.3313	1.1155	0.9599	0.951	0.6985	0.698	0.6818	0.5517	1.2007	0.5704	0.9983	1.4263
d_{56}	0.7678	1.8221	1.0558	1.24	1.114	1.0776	1.0428	0.9157	1.185	0.875	0.6219	0.4465	0.7024	1.3246	0.8798	0.9055	1.497	1.1657	1.7128	1.5542
d_{57}	1.3522	1.1026	1.2646	1.4084	1.193	1.2196	0.3	0.7468	1.3679	0.9923	1.2408	0.8776	1.1646	1.467	0.9421	0.9941	1.0436	1.311	0.9461	0.7801
d_{58}	1.0473	1.222	0.9231	1.0211	0.7715	0.7016	1.0711	1.3802	0.8666	0.4313	0.9435	0.7345	0.9138	0.6789	0.4285	0.5105	1.4295	0.544	1.1904	0.8122
d_{59}	0.8884	1.294	0.8228	1.0116	1.531	0.8956	1.6188	1.4583	0.7893	1.1671	0.8761	0.9817	0.7316	0.8865	1.1038	0.9842	0.7163	0.7581	1.43	1.5263

Table 6.3: Anchor Point Analysis for painting: Nevermore, Paul Gauguin (1897)

Appendix B

```
/**
   Shader 1: Minimum distance calculator
*/
precision highp float;
varying vec2 vTextureCoord;
uniform sampler2D tRGBA;        //target RGBA texture
uniform sampler2D dRGBA;        //database RGBA texture
uniform int maxNoFeatures;

void main() {
    float step = 1.0/float(maxNoFeatures);
    vec4 targetRGBA = texture2D(tRGBA, vec2(vTextureCoord.s, 0.5));
    int i = 0;
    vec2 dTexels = vTextureCoord;
    vec3 db = vec3(1.0,1.0,1.0);

    //min distance found
    float bMin = 10.0;
    float b = 0.0;
    float angleDiff = 0.0;
    vec3 target = targetRGBA.rgb * 2.0 - vec3(1.0,1.0,1.0);
    float epsilon = 1.0/(2.0*float(maxNoFeatures));

    //best matching feature point
    float bD = 0.0;
    while(i < maxNoFeatures) {
```

cont-

-cont

```
         dTexels = vec2(step*float(i)+epsilon, vTextureCoord.t);
         db = texture2D(dRGBA, dTexels).rgb * 2.0 - vec3(1.0,1.0,1.0);

         angleDiff = abs(texture2D(dRGBA, dTexels).a - targetRGBA.a);

         //distance calculation for each pair of points
         b = b(db, target)  + (1.0-2.0*abs(angleDiff-0.5))*0.15;
         if(b<bMin)  {
            bMin = b;
            //keeping tracking of which point matched best
            bD = step*float(i);
         }
         i++;
    }
    //to encode between 0 and 1, divide by sqrt(12) + 0.15 = 3.614
    //output min distance for each target object point
    gl_FragColor = vec4(bMin/3.614, bD, 1.0, 1.0);
}
```

```
/**
   Shader 2: Scale, rotation and anchor points calculator
*/
precision highp float;
varying vec2 vTextureCoord;
uniform sampler2D shader1output;            //output from shader 1
uniform sampler2D tXY;                      //target XY texture
uniform sampler2D dXY;                      //database XY texture
uniform int maxNoFeatures;
uniform int screenWidth;
uniform int screenHeight;

void main() {
    vec2 screenResf = vec2(float(screenWidth), float(screenHeight));

    float step = 1.0/float(maxNoFeatures);
    int i = 0;
    vec2 readTexCoord;

    //selecting anchor points
    float secondLowestDistance = 10.0;
    float lowestDistance = 10.0;

    float firstTargetAnchor = 0.0;
    float secondTargetAnchor = 0.0;
    float firstDBAnchor = 0.0;
    float secondDBAnchor = 0.0;

    float currDistanceVal = 0.0;
    float lowestDistanceFeatureNo = 0.0;

    float epsilon = 1.0/(2.0*float(maxNoFeatures));
    float featureStrength;

    while(i < maxNoFeatures) {
        readTexCoord = texture2D(shader1output, vec2((step*float(i))+
                                 epsilon, vTextureCoord.t)).rg;
        featureStrength = texture2D(dXY, vec2((step*float(i))+
                                 epsilon, vTextureCoord.t)).b;
        currDistanceVal = (readTexCoord.r * 3.614)/
                                 pow(featureStrength, 2.0);
        lowestDistanceFeatureNo = readTexCoord.g;
```

cont-

-cont

```
        if(currDistanceVal < lowestDistance) {
            secondLowestDistance = lowestDistance;
            lowestDistance = currDistanceVal;

            secondDBAnchor = firstDBAnchor;o
            firstDBAnchor = lowestDistanceFeatureNo;

            secondTargetAnchor = firstTargetAnchor;
            firstTargetAnchor = float(i)*step;
        }
        else if(currDistanceVal < secondLowestDistance) {
            secondLowestDistance = currDistanceVal;
            secondDBAnchor = lowestDistanceFeatureNo;
            secondTargetAnchor = float(i)*step;
        }
        i++;
    }

    //using firstTargetAnchor, firstDBAnchor, secondTargetAnchor,
    //secondDBAnchor we look up
    vec2 dbAnchor1 = texture2D(dXY, vec2(firstDBAnchor+epsilon,
                                vTextureCoord.t)).st * screenResf;
    vec2 dbAnchor2 = texture2D(dXY, vec2(secondDBAnchor+epsilon,
                                vTextureCoord.t)).st * screenResf;

    vec2 targetAnchor1 = texture2D(tXY, vec2(firstTargetAnchor +
                                   epsilon, 0.5)).st * screenResf;
    vec2 targetAnchor2 = texture2D(tXY, vec2(secondTargetAnchor +
                                   epsilon, 0.5)).st * screenResf;

    float d1 = distance(targetAnchor1.st, targetAnchor2.st);
    float d2 = distance(dbAnchor1.st, dbAnchor2.st);

    //determining scale
    float scale = d2/d1;

    float y2Minusy1Target = targetAnchor2.t - targetAnchor1.t;
    float x2Minusx1Target = targetAnchor2.s - targetAnchor1.s;

    float thetaTarget = atan(y2Minusy1Target, x2Minusx1Target);

    float y2Minusy1DB = dbAnchor2.t - dbAnchor1.t;
```

cont-

-cont

```
    float x2Minusx1DB = dbAnchor2.s - dbAnchor1.s;
    float thetaDB = atan(y2Minusy1DB, x2Minusx1DB);

    //determing rotation
    float rotation = mod(thetaDB - thetaTarget + 6.283185, 6.283185);

    //default values used if dbAnchor1 == dbAnchor2
    if(d2==0.0) {
        scale=1.0;
        rotation=0.0;
    }
    //encode scale and rotation between 0 and 1: divide by 16 and 2pi
    //output scale, rotation and anchor point
    gl_FragColor = vec4(firstDBAnchor, firstTargetAnchor,
                                   scale*0.0625, rotation*0.159);
}
```

```
/**
   Shader 3: Overall match estimate calculator
*/
precision highp float;
varying vec2 vTextureCoord;
uniform int maxNoFeatures;
uniform sampler2D tRGBA;                    //target XY texture
uniform sampler2D dRGBA;                    //database XY texture
uniform sampler2D scaleAndRotation;         //output from shader 2
uniform sampler2D tXY;                      //target XY texture
uniform sampler2D dXY;                      //database XY texture
uniform float clamp;
uniform vec2 biasWeight;
uniform int screenWidth;
uniform int screenHeight;

void main() {
    vec2 screenResf = vec2(float(screenWidth), float(screenHeight));
    float step = 1.0/float(maxNoFeatures);

    int d = 0;
    vec4 targetRGBA;
    vec4 targetXY;
    vec4 dRGBA;
    vec2 dXY;

    vec2 dTexels;

    float epsilon = 1.0/(2.0*float(maxNoFeatures));
    float epsilonWeight = 1.0/(2.0*10.0);

    vec4 scaleAndRotationLookup = texture2D(scaleAndRotation,
                                                       vTextureCoord);
    float dbAnchorCoord = scaleAndRotationLookup.r;
    float targetAnchorCoord = scaleAndRotationLookup.g;

    //looking up scale and rotation estimates
    float scale = scaleAndRotationLookup.b*16.0;
    scale = clamp(scale, 1.0/16.0, 16.0);
    float rotation = scaleAndRotationLookup.a*6.283185;

    vec2 dbAnchorXY = texture2D(dXY, vec2(dbAnchorCoord +
                          epsilon, vTextureCoord.t)).st * screenResf;
```

cont-

-cont

```
    vec2 targetAnchorXY = texture2D(tXY, vec2(targetAnchorCoord +
                          epsilon, 0.5)).st * screenResf;

    int t = 0;
    float sum;

    float scaledAndRotX;
    float scaledAndRotY;
    vec2 scaledAndRotatedScreenFP;
    vec2 originalScreenFP;

    float angleDiff = 0.0;

    float rgbaMatch = 0.0;
    float xyMatch = 0.0;

    float sumMatch = 0.0;
    float scaleBias = 0.0;
    float rotationBias = 0.0;

    float combinedMatch = 0.0;
    float minCombinedMatch = 100.0;
    float weight = vTextureCoord.s+epsilonWeight;

    while(t < maxNoFeatures) {
        minCombinedMatch = 100.0;
        originalScreenFP = texture2D(tXY, vec2(float(t) * step
                          + epsilon, 0.5)).st * screenResf;

        //transforming target point
        scaledAndRotX = dbAnchorXY.s + scale*((cos(rotation)*
                          (originalScreenFP.s - targetAnchorXY.s)) -
                          (sin(rotation)*(originalScreenFP.t -
                          targetAnchorXY.t)));
        scaledAndRotY = dbAnchorXY.t + scale*((sin(rotation)*
                          (originalScreenFP.s - targetAnchorXY.s)) +
                          (cos(rotation)*(originalScreenFP.t -
                          targetAnchorXY.t)));
        scaledAndRotatedScreenFP = vec2(scaledAndRotX, scaledAndRotY);


        //decode both the lookups :  rgb*2.0 - vec3(1.0,1.0,1.0)
```

cont-

-cont

```
        //handle angles differently
        targetRGBA = texture2D(tRGBA, vec2((float(t)*step)+epsilon,
                                                            0.5));
        targetRGBA.rgb = targetRGBA.rgb*2.0 - vec3(1.0,1.0,1.0);

        d = 0;
        while(d < maxNoFeatures){
            dTexels = vec2(step*float(d) + epsilon, vTextureCoord.t);
            dXY = texture2D(dXY, dTexels).st * screenResf;

            dRGBA = texture2D(dRGBA, dTexels);
            dRGBA.rgb = dRGBA.rgb*2.0 - vec3(1.0,1.0,1.0);

            angleDiff = abs(dRGBA.a - mod(targetRGBA.a +
                          scaleAndRotationLookup.a, 1.0));
            //calculating RGBtheta match
            rgbaMatch = distance(dRGBA.rgb, targetRGBA.rgb) +
                          (1.0 - 2.0 * abs(angleDiff-0.5))*0.25;

            //calculating XY match
            xyMatch =  distance(scaledAndRotatedScreenFP, dXY.st)/
                                                     screenResf.s;
            //max possible values: 1.41 and 3.71
            //combining colour change and xy match values
            combinedMatch = ((1.0-weight)*xyMatch) +
                                         (weight*rgbaMatch);
            combinedMatch = clamp(combinedMatch, 0.0, clamp);
            minCombinedMatch = min(combinedMatch, minCombinedMatch);
            d++;
        }

        //biasing towards no scale and no rotation
        scaleBias = (abs(log2(scale))/4.0) * biasWeight.s;
        rotationBias = (1.0-2.0*abs(scaleAndRotationLookup.a - 0.5))
                                                    * biasWeight.t;
        //summing across all points in the target object
        sumMatch = sumMatch + minCombinedMatch;
        t++;
    }
    //encode scale and rotation between 0 and 1: divide by 16 and 2pi
    //output overall match value
    gl_FragColor = vec4(scale*0.0625, (sumMatch + scaleBias+
        rotationBias) /float(maxNoFeatures), weight, rotation*0.159);
}
```

References

Bay, H., Tuytelaars, T., & Gool, L. V. (2006). Surf: Speeded up robust features. In *In eccv* (pp. 404–417).

Bradski, G., & Kaehler, A. (2008). *Learning opencv: Computer vision with the opencv library.* O'Reilly Media. Retrieved from http://books.google.co.nz/books?id=seAgiOfu2EIC

Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). Brief: Binary robust independent elementary features. In *Proceedings of the 11th european conference on computer vision: Part iv* (pp. 778–792). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=1888089.1888148

Chen, C. (2014). *Computer vision in medical imaging.* World Scientific. Retrieved from https://books.google.co.nz/books?id=K-XqmQEACAAJ

Chen, H.-T., Chang, H.-W., & Liu, T.-L. (2005). Local discriminant embedding and its variants. In *Proceedings of the 2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05) - volume 2 - volume 02* (pp. 846–853). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dx.doi.org/10.1109/CVPR.2005.216 doi: 10.1109/CVPR.2005.216

Computer vision and the future of mobile devices. (n.d.).

Darpa urban challenge. (n.d.).

Ensor, A., & Hall, S. (2011). Gpu-based image analysis on mobile devices. *CoRR*, *abs/1112.3110*. Retrieved from `http://dblp.uni-trier.de/db/journals/corr/corr1112.html#abs-1112-3110`

Ensor, A., & Hall, S. (2013, Nov). Colourfast: Gpu-based feature point detection and tracking on mobile devices. In *Image and vision computing new zealand (ivcnz), 2013 28th international conference of* (p. 124-129). doi: 10.1109/IVCNZ.2013.6727003

Fei-Fei, L., Fergus, R., & Perona, P. (2004, June). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *Computer vision and pattern recognition workshop, 2004. cvprw '04. conference on* (p. 178-178). doi: 10.1109/CVPR.2004.109

Freund, Y., & Schapire, R. E. (1997, August). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, *55*(1), 119–139. Retrieved from `http://dx.doi.org/10.1006/jcss.1997.1504` doi: 10.1006/jcss.1997.1504

Fung, J., & Mann, S. (2004, May). Computer vision signal processing on graphics processing units. In *Acoustics, speech, and signal processing, 2004. proceedings. (icassp '04). ieee international conference on* (Vol. 5, p. V-93-6 vol.5). doi: 10.1109/ICASSP.2004.1327055

Fung, J., & Mann, S. (2005). Openvidia: Parallel gpu computer vision. In *Proceedings of the 13th annual acm international conference on multimedia* (pp. 849–852). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1101149.1101334` doi: 10.1145/1101149.1101334

Ganster, H., Pinz, A., Rhrer, R., Wildling, E., Binder, M., & Kittler, H. (2001). Automated melanoma recognition. *IEEE Transactions on Medical Imaging*, *20*, 233–239.

Garcia-garrido, M. A., Ocaa, M., Llorca, D. F., Arroyo, E., Pozuelo, J., & Gavilan, M. (2012). *Complete vision-based traffic sign recognition supported by an i2v communication system.*

Gartner says smartphone sales accounted for 55 percent of overall mobile phone sales in third quarter of 2013. (n.d.).

Gehrig, S., & Stein, F. (1999). Dead reckoning and cartography using stereo vision for an autonomous car. In *Intelligent robots and systems, 1999. iros '99. proceedings. 1999 ieee/rsj international conference on* (Vol. 3, p. 1507-1512 vol.3). doi: 10.1109/IROS.1999.811692

Gpu: Changes everything. (n.d.).

Grana, C., Pellacani, G., Cucchiara, R., & Seidenari, S. (2003, Aug). A new algorithm for border description of polarized light surface microscopic images of pigmented skin lesions. *Medical Imaging, IEEE Transactions on*, *22*(8), 959-964. doi: 10.1109/TMI.2003.815901

Grzymala-Busse, P., Grzymala-Busse, J., & Hippe, Z. (2001). Melanoma prediction using data mining system lers. In *Computer software and applications conference, 2001. compsac 2001. 25th annual international* (p. 615-620). doi: 10.1109/CMPSAC.2001.960676

Hall, S. (2014). *Gpu accelerated feature algorithms for mobile devices* (Unpublished doctoral dissertation). Auckland University of Technology.

Harris, C., & Stephens, M. (1988). A combined corner and edge detector. In *In proc. of fourth alvey vision conference* (pp. 147–151).

Huang, T. (1996). Computer vision: Evolution and promise.. Retrieved from `http://cds.cern.ch/record/400313/files/p21.pdf`

Ke, Y., & Sukthankar, R. (2004, June). Pca-sift: a more distinctive representation for local image descriptors. In *Computer vision and pattern recognition, 2004. cvpr 2004. proceedings of the 2004 ieee computer society conference on* (Vol. 2, p. II-506-II-513 Vol.2). doi: 10.1109/CVPR.2004.1315206

Kim, J., Park, E., Cui, X., Kim, H., & Gruver, W. (2009, Oct). A fast feature extraction in object recognition using parallel processing on cpu and gpu. In *Systems, man and cybernetics, 2009. smc 2009. ieee international conference on* (p. 3842-3847). doi: 10.1109/ICSMC.2009.5346612

Krogh, B., & Thorpe, C. (1986, Apr). Integrated path planning and dynamic steering control for autonomous vehicles. In *Robotics and automation. proceedings. 1986 ieee international conference on* (Vol. 3, p. 1664-1669). doi: 10.1109/ROBOT.1986.1087444

Kruger, J., & Westermann, R. (2003). Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics*, *22*, 908–916.

Lee, R., Kitayama, D., Kwon, Y.-J., & Sumiya, K. (2009). Interoperable augmented web browsing for exploring virtual media in real space. In *Proceedings of the 2nd international workshop on location and the web* (pp. 7:1–7:4). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1507136.1507143` doi: 10.1145/1507136.1507143

Lefevre, E., Colot, O., Vannoorenberghe, P., & de Brucq, D. (2000). Knowledge modeling methods in the framework of evidence theory: an experimental comparison for melanoma detection. In *Systems, man, and cybernetics, 2000 ieee international conference on* (Vol. 4, p. 2806-2811 vol.4). doi: 10.1109/ICSMC.2000.884422

Liepe, J., Barnes, C. P., Cule, E., Erguler, K., Kirk, P. D. W., Toni, T., & Stumpf, M. P. H. (2010). Abc-sysbio - approximate bayesian computation in python with gpu support. *Bioinformatics*, *26*(14), 1797-1799. Retrieved from `http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics26.html#LiepeBCEKTS10`

Lindeberg, T. (1998, November). Feature detection with automatic scale selection. *Int. J. Comput. Vision*, *30*(2), 79–116. Retrieved from `http://dx.doi.org/10.1023/A:1008045108935` doi: 10.1023/A:1008045108935

Lowe, D. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. the proceedings of the seventh ieee international conference on* (Vol. 2, p. 1150-1157 vol.2). doi: 10.1109/ICCV.1999.790410

Miccai workshop on medical computer vision: Algorithms for big data (bigmcv). (n.d.).

Mikolajczyk, K., & Schmid, C. (2001). Indexing based on scale invariant interest points. In *Computer vision, 2001. iccv 2001. proceedings. eighth ieee international conference on* (Vol. 1, p. 525-531 vol.1). doi:

10.1109/ICCV.2001.937561

Mikolajczyk, K., & Schmid, C. (2002). An affine invariant interest point detector. In *Proceedings of the 7th european conference on computer vision-part i* (pp. 128–142). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=645315.649184

Mikolajczyk, K., & Schmid, C. (2005, Oct). A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *27*(10), 1615-1630. doi: 10.1109/TPAMI.2005.188

Public road urban driverless-car test 2013. (n.d.).

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning* (pp. 873–880). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1553374.1553486 doi: 10.1145/1553374.1553486

Reitmayr, G., & Schmalstieg, D. (2003). Location based applications for mobile augmented reality. In *Proceedings of the fourth australasian user interface conference on user interfaces 2003 - volume 18* (pp. 65–73). Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=820086.820103

Rendering pipeline overview. (n.d.).

Robotics research group. (n.d.).

Rosten, E., & Drummond, T. (2005). Fusing points and lines for high performance tracking. In *In international conference on computer vision* (pp. 1508–1515). Springer.

Search for pictures with google goggles. (n.d.).

Sonka, M., Hlavac, V., & Boyle, R. (2007). *Image processing, analysis, and machine vision.* Thomson-Engineering.

Szeliski, R. (2010). *Computer vision: Algorithms and applications* (1st ed.). New York, NY, USA: Springer-Verlag New York, Inc.

Tommasi, T., Torre, E. L., & Caputo, B. (2006). *Melanoma recognition* *using representative and discriminative kernel classifiers.*

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In (pp. 511–518).

Wang, X. (2007, May). Laplacian operator-based edge detectors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *29*(5), 886-890. doi: 10.1109/TPAMI.2007.1027

Want, R. (2010, July). iphone: Smarter than the average phone. *Pervasive Computing, IEEE*, *9*(3), 6-9. doi: 10.1109/MPRV.2010.62

What is gpu computing? (n.d.).

Yuv pixel formats. (n.d.).