An Interactive Creative environment through dynamically changing Augmented Reality Landscapes

Terence Tse
Supervisor: Benjamin Glocker
Second Marker: Abhijeet Ghosh

June 2, 2015

Contents

1 DO Abstract

Oh

2 DO Introduction

Creativity involves breaking out of established patterns in order to look at things in a different way. *Edward de Bono.* I think it's fair to say that personal computers have become the most empowering tool we've ever created. They're tools of communication, tools of creativity, and they can be shaped by their user. *Bill Gates.*

Augmented Reality(AR) has come into the limelight once again with the recent release of the Google Glass. AR has been around for a while now, most of the time being thought of the future of technology but with applications ending up with limited use, less than satisfying performance and being gimmicky, it has been hanging around in the background for a while.

AR in a nutshell is reality, augmented! A view of the real world is presented to the user but it has been enriched in some sort of way. If you are unaware of what looking through a Google Glass is like, one pretty well known example is the film Terminator. When seeing from Terminator's viewpoint, we are presented with a red screen, this is what a normal person would see but in red. However, on the screen, it has a crosshair, which lock onto a person, their statistics in the right hand corner, when it's your target, the words flash in view. This is what augmented reality is.

Of course, we're not all cyborg assassins, hence why things like the Google Glass have been created. Our eyes already give us a lot of information about the world, but having even more information presented to us than what we naturally have is Augmented Reality. Already, anyone with a semi-decent mobile phone can play around and take benefits from Augmented Reality, it does not take large amounts of processing power to produce an AR image. The key is that AR is real time, our view gets updated as we see it.

Augmented Reality is not just limited to some head mounted device. In addition to our phones, we can use tablets or just plain computers in which to play with AR. All we need is a camera which can capture the world and then allow us to view it either with additional information or even diminished information! One use case is in the British Maritime museum, where visitors can aim cameras at special AR exhibits and have the item explained to them visually. The range applications of AR are overwhelming, imagine a Dinosaur coming to life in a History museum and you can see it navigating the halls. Imagine having an AR mirror that superimposes clothes onto your body that fit, without having to go and search and try on the clothes themselves. Imagine being on a construction project and you are able to view the outcome on your tablet, even if the old structure has not been cleared away for the new project to begin! Augmented Reality can do all of that already and the scope of new applications is, no pun intended, as far as the eye can see!

As previously stated, AR has many applications in terms of games and entertainment. These come in all different styles, from simple use of a camera, to using fixed feature points as markers, or using patterns as feature points to identify placement of objects in the scene. Right now, these feature points are either very simple or immutable. They are something that the computer can directly identify which is obviously good for performance reasons. However, what about the ability to create your own feature points on the fly? If we able to immediately draw feature points and by extraction of these points, augment reality based off of these points, it would greatly improve the experience of Augmented Reality, removing the need to create the marker on a computer and just, for example, grabbing a piece of paper, drawing it yourself and have it work straight away! I shall refer to this as a feature drawing throughout the rest of the report.

However, AR may also be used for more commercial and other practical uses. When people think of AR, it usually defaults to thinking about games and indeed there is a lot of future in that industry with AR, there are also Educational uses, Project and Landscape Planning, Commerce and Animation, the list goes on. The problem right now is that simple AR applications which augment a scene with, say a dinosaur or some other object, those generated images are still quite obviously generated by a computer, especially when they try to get more complex. In a bright room, you may place furniture inside which looks semi-realistic, it has a computed shadow and gradient change. However, an ideal situation is when we augment reality and it doesn't even feel as if anything is different.

Throuhg this project, I hope to make some headway into providing creative individuals a program that allows them to draw out their dreamt up landscapes and have it rendered before them in 3D. The project will be focused on mainly forming land features, namely hills; however, the general principal can be transferred over to more practical use cases by introducing other forms of computer vision such as template matching. This will serve as a basis for a creative environment for users, requiring minimal interaction with the user and more time for pen on paper work.

2.1 Project Goals

The bigger picture and where this project hopes it can contribute and make headway into is permitting a dynamic and responsive AR application that allows creators to quickly draw up environments which can then be visualised. Environments can also be changed in real time and will be reflected in the AR without need for additional user interaction. The

completion of this task will have numerous transferable benefits.
However, the amount of work that this goal encompasses could be infintissimal depending on what kind of features would be deemed necessary, desirable or just an extension. There are also numerous parts of a project like this could be analysed in much further detail. As a result, the objectives for this project are:

- Create a functioning Augmented Reality program which takes a video stream and overlays a 3D landscape on a contour map created by the user.
- Using a method of video and scene tracking/recognition, allow users to make changes to their contour map and have the landscape update in real time. This essentially allows dynamically changing Augmented Reality landscapes.
- Perform an analysis on the performance and responsiveness of the final product of this project. This will be done through user testing.

3 Background

3.1 Augmented Reality

What is Augmented Reality(AR) and why should we be interested in it?

In recent years, it becomes harder and harder to concisely describe what constitutes as Augmented Reality. The reason for this being that as technology improves more and more things are being achieve that allow users to extend their perceived reality with extra data.

Simply put, Augmented Reality is when the environment around you can be extneded by combining both real and virtual data components. Azuma's paper[3], *A Survey of Augmented Reality*, describes AR as having the following three characteristics:

1. Combines Real and Virtual (data)
2. Interactive in real time
3. Registered in 3D

While these are the basic things that constitute AR, there are three ways, outlined in Mackay's paper[8], *Augmented reality: linking real and virtual worlds: a new paradigm for interacting with computers*, in which you can augment reality.
These are:

1. Augmenting the User
2. Augmenting Physical devices/objects
3. Augment the Environment

3.1.1 Augment the User

Augmenting the user is having the user carrying, wearing or using a device that provide the user with extra information. When we talk about augmenting the user, the devices we talk about are usually heads-up displays (HUDS) which the user will have a screen in front of their eyes in some shape or form. One example of this is Google Glass, allows users to project information about what they see or their GPS location onto parts of the glass. If the user was looking at the Eiffel Tower through their glass, with GPS coordinates and a scan of the shape of the Eiffel Tower, Google Glass would be able to tell that what the user was looking at. From there, it can project information about the landmark to the user which would not have been accessible so easily otherwise!

This is one such trivial application of user worn AR, it is very useful for lesser known things, for example, instead of looking at a famous landmark, the user could be an engineer, looking at a complex piece of machinery and a HUD allowing them to get information on each component. This is also being trialled in Medicine, where doctors can use Augmented Reality to help teach surgical procedures or even help them in analysis of medical images such as X-Rays.

3.1.2 Augment Devices

When an object or device has a small computer placed into it, it can be considered as an augmented device. One example of this is again in surgery and medical environments. Having tools such as scalpels or other medical equipment fitted with computers to plot out how much of the skin has to be cut, for example, allows the user to receive much more information than they would be able to than without. Mackay also talks about these briefly in her paper.

3.1.3 Augment the Environment

This form or AR doesn't add any additional hardware to devices nor the user. It focuses on having external devices, such as projectors or cameras take input from the user and use that to transofrm the environment that they are in or are manipulating. This class of AR is much wider than the two aformentionned as there are less things imposed on an object/person which allow a larger spectrum of devices to be used.

An example of Augmenting the environment is an AR keyboard, a projector displays a keyboard in front of where it is placed, the user can then "press" keys and the kyboard will be able to detect which key has been selected and translate it into computer input.

A more common device is your smart phone! Tablet or even computer screens can be included in this form of AR. There are plenty of mobile applications that allow users to experience AR in their own unique way. An example used in the British Maritime Museum is the ability to take a tablet with predownloaded apps that allow users to engage in an AR game. Through the game, which is primarily aimed at children, allows them to learn about the exhibits in a more fun and interactive way. There are also similar applications in retail, for example in retail! Augmented reality mirrors are being used to avoid the time in the changing room and allow customers to "try" on pieces of clothing through AR by overlaying the garment over their captured video.

Another fun implementation of AR is the Augmented Sandbox. A projector and a camera are used to transform a sandbox into a remoldable landscape. The camera uses depth

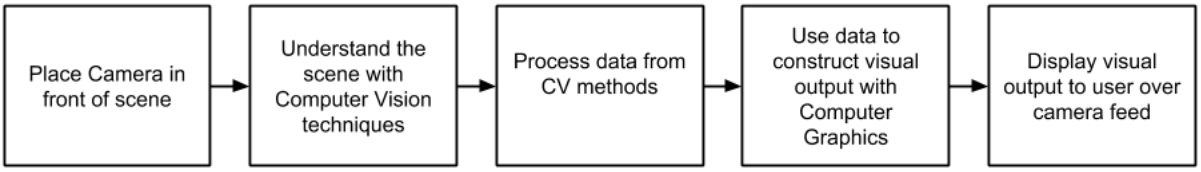


Figure 1: Augmented Reality pipeline.

sensing and the input from here is transformed into lanscape height and the projector correspondingly projects land or water onto these areas. The magic happens when the sand is manipulated and the whole system re-renders the landscape in real time to provide an interactive augmented environment! Sarah Reed from the University of California lead the authoring of the persentation about this piece of tehnology[12].

3.2 Construction of AR scenes.

Augmented Reality has two main parts to it; Computer Vision and Computer Graphics. There may also be all sort of other computer science topics melded in, for example Machine Learning, Artificial Intelligence and various others. The pipeline in Figure 1 describes how a basic AR application should be created.

The middle stage, where the data from Computer Vision methods is processed is where we can usually slot in thing slike Machine Learning or other kinds of data manipulation, before handing it over to be rendered by Computer Graphics applications. For the purposes of this project, I will not need, nor do I intended, to stray from this basic pipeline where data processing in the middle will be minimal.

3.3 Computer Vision and Image Processing

A key part of this project is the Computer Vision aspect. The field of Computer Vision involves the understanding of data from images and pictures. "Understanding" is a broad term that encompasses many ways of manipulating or processing images in order to extract features or points of interest that can be then used by the computer.

3.3.1 Edge Detection

In an image, an edge if a set of pixels that separates to disjoint regions.

A key part of Computer Vision is line and edge detection. Edge detection is the very beginning of how we can start to identify different objects within an image. In addition,

by stripping an image down to its comprising edges, we greatly decrease the amount of information in the image whilst preserving the important parts which we want to operate on. As a result, how edges are detected have a large contribution to what we can do with certain images.
Edge detection tries to find the set of pixels that separate two disjoint areas of an image by analysing sets of neighbouring pixels and looking for discontinuities in intensity or texture. A variety of factors can cause these (local) discontinuities:

1. Object colour and texture. If the object has any change in colour or texture, then where the change occurs, an edge will be detected.

2. A change of surface normal. In the real world, we have 3D objects, as a result, there will be a surface normal change when we consider, say, the top face of a box against the side face. There is an edge between the two and this is partly also due to the next factor.

3. Illumination. A change of illumination or simply due to some part of the obejct in question casting a shadow will cause some regions of the image to be darker or lighter in intensity, even if said regions are part of the same object. Edges will be detected in this instance.

4. Two objects. This a large part of why we even do edge detection, to detect objects. If there are two objects, then, likely, there will be a difference in colour and intensities of their constituant pixels. However, this is not always the case and also as can be reasoned from above, these regions may belong to the same multi-coloured object.

When it comes to
We can distinguish edge detectors into two sets, the Laplacian of Gaussain detectors and the Gradient-based detectors.

3.3.2 Gradient-based detectors

Gradient edge detectors look for changes of gradient (first-order) of neighbouring pixels. Some common gradient-based detectors are Sobel, Prewitt, Canny and Robert. Each of these detectors starts off with an image and an x and y kernel. The kernels help determine the gradient change and y directions of the image, different kernels (size and values) are what separate the different Gaussian-based detectors. The final gradient magnitude for a given pixel is then the square root of the sum of these:

$$G = \sqrt{\Delta x^2 + \Delta y^2} \tag{1}$$

Users can then set a threshold value for which any pixel with a final gradient that is above that threshold will be treated and marked as an edge, the others will be discarded.

3.3.3 Laplacian (of Gaussian)-based detectors

Laplacian based detectors are similar to Gradient-based ones except that instead of taking the first derivative, they consider the second derivative. Their goal is to look for zero-crossings, i.e. local maxima in gradient changes (i.e. looking for when gradient starts going negative). The change of gradient sign, which occurs when second order derivative is 0, indicates that the intensity of the observed pixels begins changing also. This method can be coupled with a Gaussian kernel which serves to de-noise the image by smoothing out the pixel intensities.

3.3.4 Canny Edge Detector

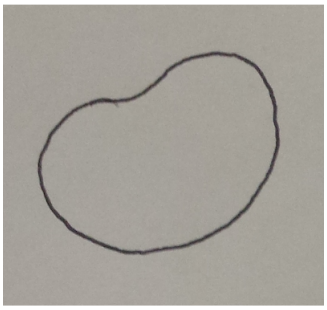
There have been numerous methods of detecting edges put forward, some Gradient-based and Laplacian based edge detection methods are outlined in *A Comparison of various Edge Detection Techniques used in Image Processing* [14]. However, the most popular edge detector out there is the Canny Edge detector, put forward by John Canny in his paper *A Computational Appraoch to Edge Detection*[5]. The paper dicusses a methematical approach to edge and ridge detection which is then analysed on in Ding and Goshtasby's paper, *On the Canny Edge Detector*[6]. Rashmi's paper, *Algorithm and Technique on various Edge Detection: A Survey* [11] compares various edge detection algorithms and finds that the Canny edge detector comes out on top, with Maini's paper *Study and Comparison of Various Image Edge Detection Techniques* [9] showing the same result.
The Canny Edge detector falls within the Gaussian-based Edge detectors and can be described algorithmically below:

- Apply a Gaussian filter over the image to remove noise by smoothing
- Take the Gradient of the image pixels over each direction, just as would be done in a normal Gaussian-based detector.
- Apply non-maximum suppression such that only local maxima are marked as edges (the pixel has a greater gradient magnitude than all of its neighbours) resulting in a thin edge. The thinning of found edges is to remove some affect of the blur. After this, perform double thresholding, those higher than the threshold are "strong" edges and those under are weak edges.
- Another round of removing edges is applied by using hysteresis. This involves identifying "strong" edges,and then removing all edges that are not connected directly to these strong edges.

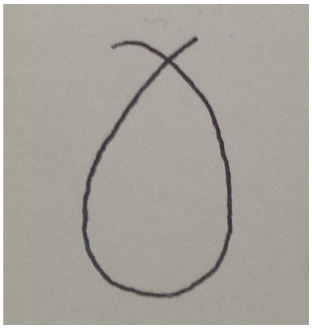
The steps are more fully explained in the paper by 09gr820, *Canny Edge Detection*[1].

3.3.5 Open and Closed contours

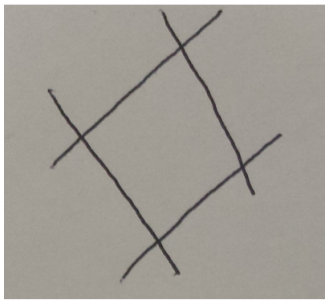
We understand a closed contour to be a loop. By this we mean that if we had to draw a closed contour, we begin at a point on a piece of paper and end at that point, either by not removing our pen from the page or, if done so, will resume at the same point to finish up the contour. The existence of white space between points on a contour mean that the contour is open, thus not closed. Examples of open and closed contours can be seen in Figure 2 and Figure 3.



(a) Contour is fully closed, no gaps

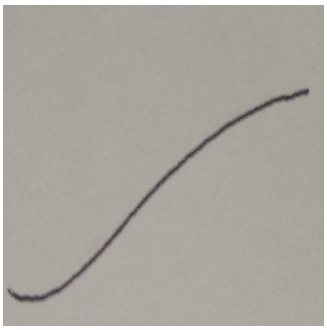


(b) Contour is closed, but with artefacts on the ends

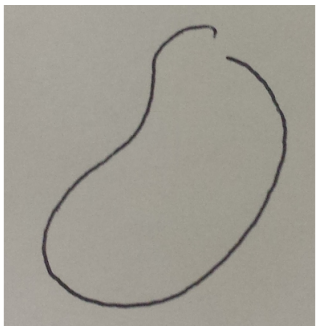


(c) This is also a closed contour

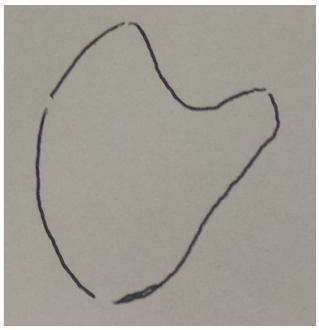
Figure 2: Examples of **closed** contours



(a) A line/edge counts as an open contour



(b) Contour open as beginning doesn't connect to end



(c) This contour is open with several gaps

Figure 3: Examples of **open** contours

For feature detection, and especially within this project, we look for *closed* contours, they provide us with the most information. An open contour may be linkeed to some kind of feature point but if it is not closed, its usefulness is very low. In particular, we want closed

contours so that any contours that are drawn within them can then lend themselves to a natural hierarchy/ordering. In the context of this project, those within closed contours will represent areas that are of higher elevation than the area contained between itself and the outer contour. When contours are not closed, it becomes harder for us to identify this hierarchy and ordering within the image, which would mean a lot more difficult computation to be done to achieve our goal of generating a corresponding landscape. Indeed, this is the problem of many similar projects.

3.3.6 Mathematical Morphology

One technique to help close contours/loops in Computer Vision is *Closing*. To understand *Closing*, we must first introduce the ideas of Mathematical Morphology (MM). MM is a method of analysing geometric objects and structures, typically used on images but can also be applied to other things such as graphs. Within MM, there are four basic operators, but for the purpose of this project, I shall only go into detail about the first three:

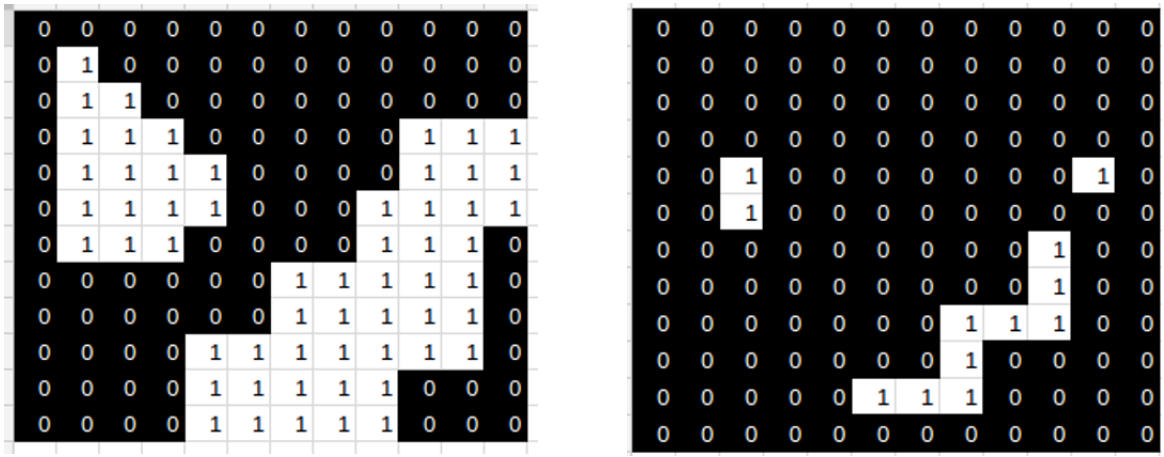
- Erosion
- Dilation
- Closing (Dilation followed by Erosion)
- Opening (Erosion followed by Dilation)

The operators were originally defined for binary images and so shall be described below in that sense, however, the functionality has been extended to include grayscale images as well, which is very useful when dealing with photographed images or video feed as will be done within this project.
For these operators, let us assume there is a binary image, with pixels holding values of either 0 or 1. In correspondence to a typical colour image, 1 indicates white, a brighter area, of the image and 0 corresponds to a black pixel. In the erosion and dilation operators, we have this binary image as input. We also have a corresponding kernel which we will convolute with the image. This kernel, sometimes referred to as a structuring element, usually takes the shape of a circle, a 3x3 square or a cross. To achieve different results from these operators the kernels can be changed, for example to a 10x10 square hich will cause a more drastic shrink or growth of bright areas. For the purpose of explanation, I will assume we are using a 3x3 kernel.

3.3.7 Erosion

Erosion causes bright areas of images to shrink.

Having the 3x3 kernel anchored at its centre square/pixel, we place align the kernel with every single pixel on the binary image by superimposing the kernel's anchor over it. Within the other kernel pixels (a 3x3 neighbourhood of the pixel in question), we set the pixel under the anchor to the minimum value in this neighbourhood. Thus, the only way a pixel can remain white in a binary image is if it starts off as white and all of its neighbours with regard to the kernel shape, are white too. If this is not the case and even 1 is 0 in its neighbourhood, the pixel considered is set to 0. The effect of this can be seen in Figure **??**. As you can see from the result, the original shapes have become much thinner and there are now 3 shapes instead of 2. When combined with dilation, it is clear how some contours can be "opened" as erosion comes as the last step. The result shown in Figure **??** shows how what could be the constituent pixels of a contour being broken down into 2 separate objects of interest, thus *opening* the contour. In addition, erosion can be used to remove small, likely unimportant, parts of images, such as noise, since any noise that isn't large enough to be a problem, would be fizzled out by the kernel, whose size can be changed to affect noise sensitivity.



(a) Original Binary Image

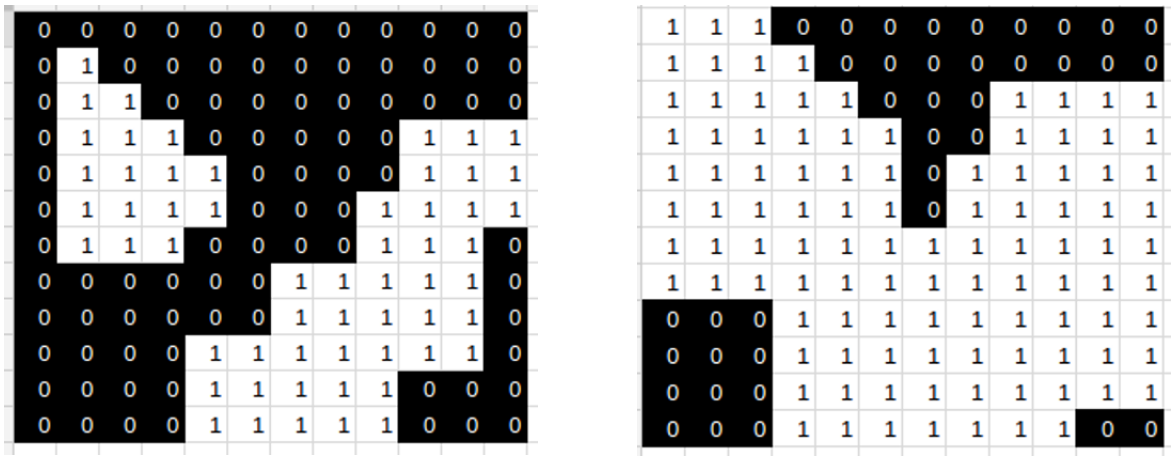
(b) Eroded Image

Figure 4: Erosion

3.3.8 Dilation

Dilation causes bright areas of image to expand.

Dilation performs the opposite of what erosion would do. We still perform convolution with our 3x3 kernel but instead we set the pixel below the kernel anchor to be the *maximum* value within the 3x3 neighbourhood. As a result, if there is even a single white pixel in the



(a) Original Binary Image

(b) Dilated Image

Figure 5: Dilation

neighbourhood, the current pixel will turn white. The only case this will not happen is if the pixel started off with a pixel value of 0 and all of its neighbours also had a value of 0. Figure 5 shows the result of dilation with the same original image as shown in the Erosion section above.

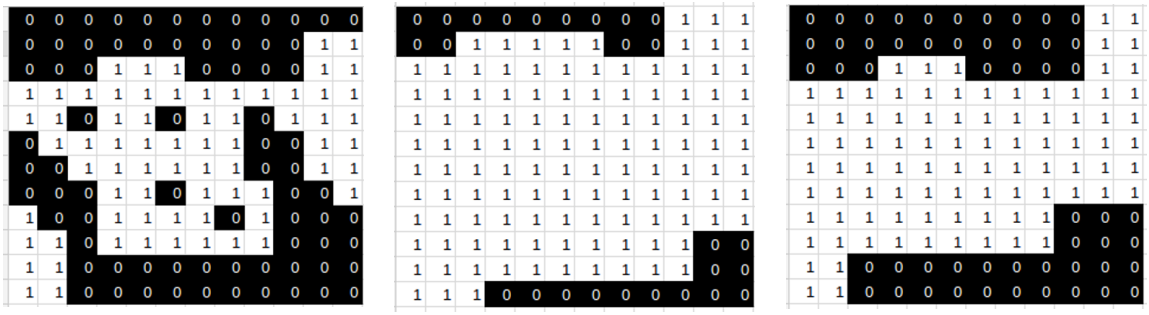
It can be observed from the result above that a dilation greatly increases the size of the bright area. In addition, if bright areas are suitably close to one another, it can cause the areas to join into one through the dilation. This is how contours with small gaps are able to be closed and will aid in the contour detection process. Again, the kernel size and shape may be changed to cover a larger area (thus join bright areas further apart) or otherwise depending on how sensitive you want the operation to be.

3.3.9 Morphological Closing

Morphological Closing is Dilation followed by Erosion.

Closing is very similar to dilation but it aims to preserve more information. The Closing operation is no different to erosion or dilation in that we provide a kernel with which to onvolute the given image with. To perform Closing, the image first has a dilation operation applied to it with the given kernel. After this, erosion is done with the same kernel on this dilated image.

By just performing dilation, we do manage to close small gaps in the image, however, every pixel is equally distorted by the kernel and so the resulting image will look quite different to the original shapes, as can be seen in Figure 5 above. In order to reduce as much of this distortion as possible, erosion is used. The overall affect of this is to make the resulting



(a) Original Picture (b) Dilation stage (c) Erosion stage

Figure 6: Closing operator step by step

image be as similar as possible to the outline of the provided original image. Closing is idempotent, after one pass the resulting image will not change from further Closings with the same kernel. The effect of closing is illustrated in Figure 6.
The original images starts off with some holes within the image. If we take this to be a contour discovered by an edge detector, then we can show how we can close it with *Closing*. After the Dilation stage, the small holes within the original image have been closed, in addition, the boundaries of the bright parts of the picture have been joined due to their proximity and the kernel shape. However, as seen in Figure 6 (b), the shape has changed a little from the original picture. After applying eroision, seen in Figure 6 (c), the final result is much closer in shape to the original picture, except it has its holes filled and the bright areas have joined. This would have closed the contour, if it were one, in one of the best ways possible, preserving most of the original shape and not losing too much fine detail through enlarging the bright area!

3.3.10 Image Differencing

For users to be able to draw their own landscapes, there has to be some way of recognising when there has been a change, physically, on paper. To automate this process will enable the program to decide for itself when there has been an update to a drawing, rather than a user having to confirm each time there is an update. Image differencing is the primary way of achieving this kind of detection. There are many different ways of actually working out whether there is a "difference" between two images and some will be further explained below. The general method for image differencing from a video feed is taking two frames, at time t_s and a time in the future t_{s+c} and compare the two.

3.3.11 Simple Intensity Differencing

The simplest form of finding the difference between two images is comparing their pixel intensities.

$$\begin{aligned} A \text{ is Image at time } t_s \\ B \text{ is Image at time } t_{s+c} \\ D \text{ is the difference Image} \\ I(pixel) \text{ is the Intensity of the supplied pixel} \end{aligned} \tag{2}$$

$$D_{i,j} = |I(A_{i,j}) - I(B_{i,j})|$$

Once the Difference Image has been computed, you can then perform a sum over all the pixels to get an indication into how much "change" there was. Depending on if the image is grayscale, binary or RGB will give different ranges of numbers, the higher the range, the less this final sum will be able to tell you about the change in the image.
A popular way to deal with this problem is to perform thresholding on the Difference Image and is explained in Rosin's Paper *Thresholding for Change Detection*[13]. Again, there are different type of thresholding such as Truncation, to-Zero, Binary and all of their inversions. For the purposes of this project, we will only look at Binary thresholding.

Binary thresholding takes a threshold value, τ, which is acquired either through statistical or empirical methods. With the threshold, any pixel value in the Difference Image that is below the threshold is effectively ignored and set to 0. The pixels above the threshold preserve their values. This method is very quick, especially when considering grayscale images where the range of pixel values is between the value 0 to 255. It begins a bit harder to use Binary thresholding with images with more colour channels.

$$D_{i,j} = \begin{cases} 0, & \text{if } D_{i,j} \leq \tau. \\ D_{i,j}, & \text{otherwise.} \end{cases} \tag{3}$$

By summing over these obtained values, the result may make more sense as it is a representation of how many pixels in the image have changed such that their change is significant. Of course this "significance" is determined by the user and the chosen τ.

It is obvious that this is avery simple way of calculating difference between images. To ensure accurate results, the camera has to stay very still otherwise, potentially, every pixel could experience a change in intensity. This can be somewhwat countered by choosing and appropriate τ. However, if looking for finer changes, the chosen τ could remove some of these changes, disabling detection.

In addition, if the images are taken in different lighting environments then that will cause inaccurate indications of difference. Take two potential scenes, exactly the same and untouchedd camera fixed. In the day time, the image captured will have a much higher intensity than an image taken in the evening. This coud cause the whole image to be above the threshold of changed but actually tell us nothing about changes in the scene, if any. The problem with dealing with illumination changes becomes more prominent in surveillance literature. A way to reduce the illumination effect is to perform a normalisation of the pixels based on average intensities and variance. This shall not be explained further but details of it and other detection techniques are highlighted it Radke's paper[10].

3.3.12 Image Ratioing

Image Ratioing is another method to compare images. The general concept is explained (along with other change detection techniques) in Singh's paper *Review Article Digital change detection techniques using remotely-sensed data*[**?**]. Again we have our two images, from before, taken at different times. From here, the pixels' intensities are compared to obtain a ratio:

$$R_{i,j} = \frac{A_{i,j}}{B_{i,j}} \tag{4}$$

The closer this $R_{i,j}$ values is to 1, the more similar the two pixels are in terms of intensity. Thus, if $R_{i,j}$ is quite far from 1, it is a good indicator that there has been change within the image. Again, thresholding can be applied to ths Ratio Image, R, which will leave only those with a significant ratio to be considered. The same steps as image differencing can then be utilised to gain a representation of the overall change in the image. The process, however, may introduce some error, especially as you consider less of the electromagnetic spectrum. For example, a grayscale image has values 0 to 255 per pixel. The ratio of a change from 3 to 1 is the same as a ratio change from 240 to 80 when the latter is clearly a larger change in the intensity of the pixel. Instead, the ratios of the pixels from several same images from different bands are used to determine the ratio with the same equation as above. The image/spectral ratioing algorithm, Singh uses in his paper utilises band 2 and band 4 images. The ratioing technique was criticised by Singh, he states:

> "The critical element of the methodology is selecting appropriate threshold values in the lower and upper tails of the distribution representing change pixel values. "

And a certain amount of empirical testing is needed to obtain good thresholding. The methods has also been criticised due to the "Non-normal distribution upon which it is

based", i.e. a Non-Gaussian distributed image which is produced as the resultof ratioing as the areas on either side of the mode are not equal, making the error rates on both sides not equal.

3.3.13 DO Camera alignment

A vast amount of Computer Vision requires the use of a camera, whether to capture video or just still pictures. However, cameras suffer from distortion in the lens and of varying degrees, usually with relation to the quality of the camera. However, since these distortions are constants, we can correct them in order to achieve undistorted images from our cameras. There are three common types of distortion:

- *Barrel Distortion*
  Where images seem more magnified the further away from the optical axis (usually a straight line through the centre of the lens) the pixel is. This causes lines to bend outward from the centre.

- *Pin Cushion Distortion*
  The opposite of Barrel Distortion, where the image becomes more magnified the closer you get to the optical axis. This type of distortion causes parts of the image to bend inward toward the centre and optical axis.

- *Moustache Distortion*
  A complex mixture of the two types of distortion, where by the image starts off seeming to be barrel distorted closer to the image centre but then turns into a Pin cushion like distortion as it gets further away.

Figure 7 shows these distortions.
When working with Computer Vision algorithms, these distortions can cause inaccuracies. As a result, ist is in our interest to find the transformations necessary to counteract these distortions. Camera distortions only need to be calculated once as it will stay constant with the camera due to its build. To undistort a camera, we can use an image with known measurements to use as a reference point to calculate the error rate of the camera, as well as store these error coefficients into a file. This file should then be supplied to any programs using the camera to correct the distortion.
From a mathematical standpoint, there are .....SEE ARUCO EXPLANATION
.....MORE (Using checkerboards)

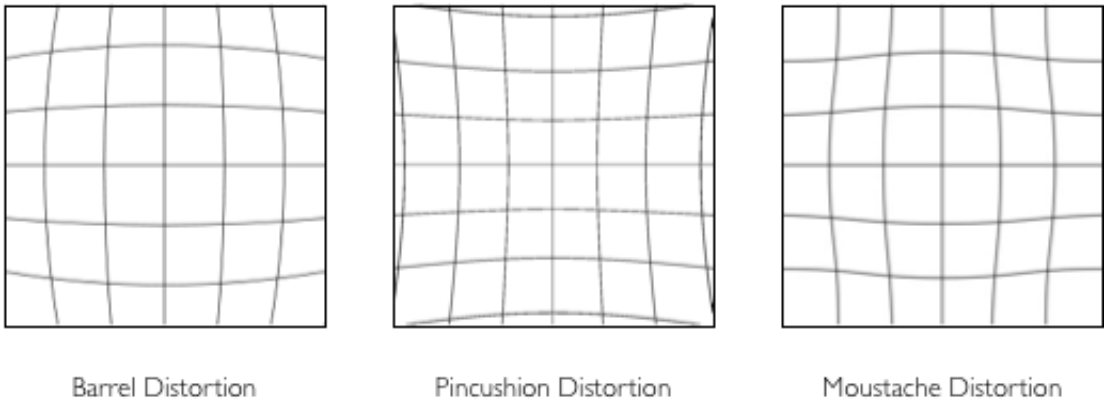


Figure 7: Lens distortion types

3.4 DO Computer Graphics

3.5 Ordnance Survey Maps

Ordnance Survey(OS) are an agency for Great Britain that specialise in making maps. The agency was formed all the way back around 1791 and has been producing maps for Britain since. Ordnance Maps are, of course, 2D and are produced on paper, long before our online maps which we use so much today. OS maps were a staple to any adventurer travelling Britain's landscape. Below in Figure 8 is a segment of a typical OS Map.
The curved lines which have numbers on them are contour lines. These contour lines dictate the height of the land above sea level at that point. All points along the contour are of equal (the stated) height. The OS map standard is that every next contour is a difference of 10m from the last. In addition, every 50m has a thicker contour. As a result, OS maps accurately depict hills, mountains and elevated land. The contours used for landscape maps in this project are heavily based off of OS map contours. A possible advanced use for this project is to turn OS maps into 3D representations. This would allow travellers who are out in the field with no internet reception for their favourite map application to simply hold view a 3D version of the OS map they should have taken on their journey. Of course, there are many other factors invovled, such as extracting other lines which may represent road of river and determining what is actually a land contour, this idea is one to keep in mind, however.

Another trait to notice aobut contour lines is that their closeness to each other also indicates how steep the slope of the area is. If contours are close, such as near the peak of the Slieve na Callaigh in Figure 8, then the land is steep. If you compare this to somewhere such as near Ballinvally, the contours are much further apart, indicating a flatter, less steeply

Figure 8: OS Map snippet

increasing/decreasing slopes.

3.6 Existing Projects

3.6.1 LandscapAR

Looking into the Augmented reality scene and looking for potential similar applications brought me across the mobile application *LandscapAR*, made by Stapps. The premise of the application is that it is aimed at graphic/landscape designers and turns drawn contour lines into an augmented reality environment you can view through your phone screen.

The application imposes several restrictions upon the user, however. The user must use a thick black pen for the app to successfully identify contours. In addition, the user must make sure that every conour is properly connected and closed. Already, this limits some things that the user is able to do, they have to take the time to be precise and also must make sure they have the correct equipment to start creating their work. In addition, the application requires the user to place white paper over a dark background/table otherwise the scene may not be recognised. Even if the paper is on a dark table, it does not guarantee the paper is identified, this may be due to illumination differences of the whole paper not being fully captured, as in Figure 9a and 9b.This is likely because they use the paper edges/corners as anchor points upon which they can align the augmented scene and use registration to align the output. In addition, it is possible for the paper to be "too far" away from the camera, when I myself tested this, it caused a good several minutes of frustration trying to align the phone camera and the paper such that the application could function seen in Figure 9c. Finally when all this is done, the user must press the "Scan" button which can cause them to lose their hard-sought alignment! An original topdown view is generated, see Figure 9d, on the first scan. After, the user can choose a free camera mode where they can move the camera around to view the scene in 3D, Figure 9e, though the ability to steadily keep this image on the screen is lackluster in some scenarios.

There are several ways that this application can be improved. For example, currently the user has to draw their contours on a piece of paper and then scan paper using their mobile device. The scanning process itself is quite cumbersome as trying to get the system to detect paper can be hit and miss. The scan analyses the image and overlays a 3D scene onto it quite well, however the downfall with this is that to make updates to the scene, ou use has to re-scan the image after addition, adding more cumbersome steps. For users who want to make several small changes here and there, this will consume a lot of time and effort on their part.

(a) The user must align their paper properly

(b) The user must align their paper properly

(c) The range of detection is limited.

(d) Output from a simple scan produces a top down image

(e) After scanning it is possible to move the camera

Figure 9: Testing LandscapAR

3.6.2 The AR Sandbox

The University of California, Davis along with the Lawrance Hall of Science and ECHO Lake Aquarium and Science centre contribute to a project called the Augmented Reality Sandbox[12]. The project allows users to turn a typical sandbox into a creative landscape

Figure 10: Children learn about watersheds with Augmented Reality.

environment, much akin the goal of this project. The system uses a Microsoft Kinect 3D camera and a projector to perform its magic. Both are positioned above the sandbox and the depth camera on the Kinect will measure the local relative depths of the sand in the sandbox. If the user has created mounds on the sanbox they will be closer to the camera, these will be converted into mountains/hills. Areas lower than a certain threshold depth will be converted into water. The input from the 3D camera will pass the data to a computer and project subsequent land/water colours onto the corresponding areas of the sandbox.

The Augmented Reality Sandbox also has a variety of other functionalities, such as creating rain over the Augmented environment by hovering a hand ofer the area of rainfall. The whole project aims to teach people, particularly children, about landscapes, watersheds and catchment areas that you can physically change, interact with, and get real-time response from. The project has gained quite a bit of interest, with alternate versions being built by others and is something that helped in the motivation for this project. The sandbox can be seen in Figure 10.

The AR Sandbox is a great tool for creativity, however it is very hard to do on a small, personal scale. Institutions and education centres can easily set up and manage such a project but the cost and maintenance is hard for an individual. In addition, the purpose is very limited and geared towards those of the younger generation to learn simple concepts

of geography. In this project, I will be using the ideas the AR Sandbox has put forward to offer an improved, alternate version which is geared towards a much larger audience, with less of a specific goal and more of an environment for creativity with less requirement for special equipment.

3.6.3 Microsoft Kinect V2

Background & History

The Microsoft Kinect(V1) was original released as an extra piece of hardware for Microsoft's Xbox360 in 2010. The Kinect for windows (V2) was released in 2012 and offered a range of improved statistics from its ancestor, V1. Since its introduction into the market, the Kinect has received a lot of attention from developers and there have been many applications for it put forward[1]. Many large companies have caught onto its capabilities and work alongside Microsoft and the Kinect.

In contrast to the Leap Motion, the Kinect V2 offers users the ability to track their whole body with reasonable accuracy. In addition, the V2 can track up to 6 individuals at one time with 25 body joints able to be placed onto the captured images. This opens up a whole new range of applications that the Kinect can be used for and this is apparent in the amount of interest in the Kinect, along with the amount of published papers and success stories of different uses of it. Some of the possible fields this device can be used include Augmented Reality, Entertainment and Games, Retail, Education and much more. The Microsoft Kinect website highlights some of these interesting projects in more detail [2].

The paper "Evaluation of the Spatial Resolution Accuracy of the Face tracking system for Kinect For Windows V1 and V2" [2] explores in more detail the increase in performance and capability of the V2 and shows that is beats V1 in every aspect.

How it works

The V2 allows us to take infrared images along with 1080p colour images. It also has depth sensors to produce depth images. Inside the V2 are 3 infrared emitters along with a colour camera, an infrared camera and some microphones. These internal components can be seen in figure 11. The microphones are along the bottom but will not be used within my project.

[1] http://www.microsoft.com/en-us/kinectforwindows/meetkinect/gallery.aspx?searchv=entertainment
[2] http://www.microsoft.com/en-us/kinectforwindows/meetkinect/default.aspx

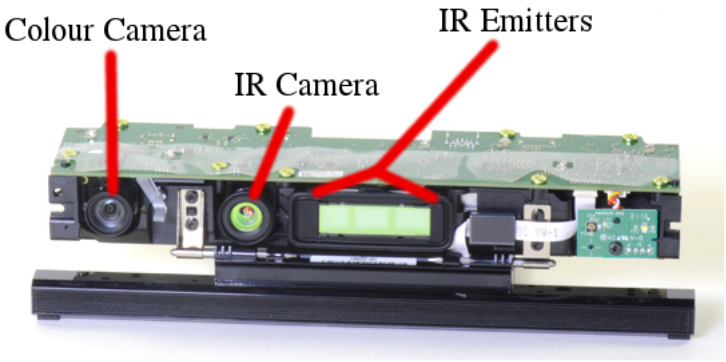


Figure 11: Inside the Kinect V2 for Windows

The Kinect V2 differs from the V1 majorly in the way it produces depth maps. V1 uses structured light while the V2 utilises Time-Of-Flight. Time-of-Flight involves emitting many short bursts of infrared light (strobing it) and then collecting it back through its camera. A very thorough discussion of how this determines the depth of objects in the captured image is present on Daniel Lau's blog [3]. The technique involves splitting a pixel in half and collecting infrared light on the pixel. However, they are on at different times, while one half is on, the other is off and depending on the ratio of how much light is collected by each half (ratio since it accounts for light absorption by objects in the scene), the relative depth of objects can be inferred.

The V2 also accounts for over exposure and saturation of pixels. This scenario occurs when there are alternative sources of infrared light, i.e. in an outdoor environment! The V2 can reset pixel values in the middle of an exposure, also explained in Daniel Lau's blog mentioned above. This then allows it to be used outside where the V1 wouldn't account for this and cause all kinds of strange behaviour; this of course means that the scope for using the V2 and its applications is significantly larger than it ancestor.

My investigations

I investigated the Kinect for Windows first hand. In figure 12 you can see an example of the depth sensing capability of V2. On the right, circled in blue, is myself. I sat around 40cm away from the camera. As I highlight in the limitations section below, this is reported as the minimum distance performance can be guaranteed by the V2. My friend, Juto Yu, is circled in yellow, he say slightly outside the view of the camera, giving me a rough indication on how wide the capture region is. The green circled object is a pillar in the central library

[3] http://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php

of Imperial College, it is definitely more than 3m away from the V2 which is reported as the range before objects become "too far". Whether it being detected is because the device can actually handle larger caption depths or due to the pillar being white and thus more reflective will need to be further looked in to.

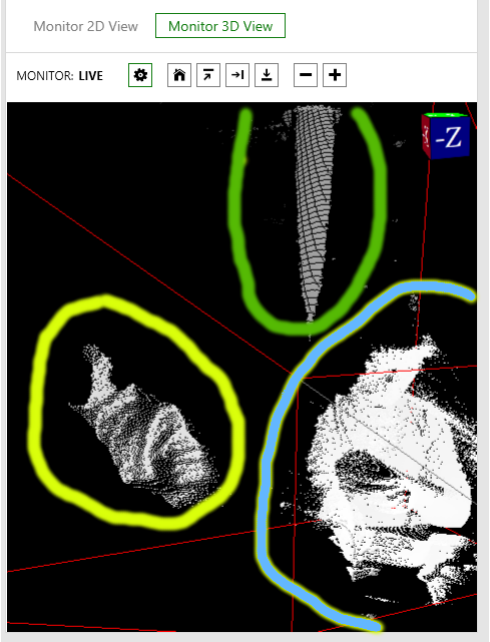


Figure 12: Depth image from Kinect V2

I also looked into the colour camera's performance. The camera has a 1080p HD video taking capability and is of a good quality. In addition, as seen in figure 13. The body tracking is pretty good and can make rather accurate implications of people's limbs even when they are sitting down and are obscured as shown. I was sitting at a desk but the V2 could make some guesses as to where my legs would be. It is also clear that multiple tracking can be done in real time at a speed that is responsive.

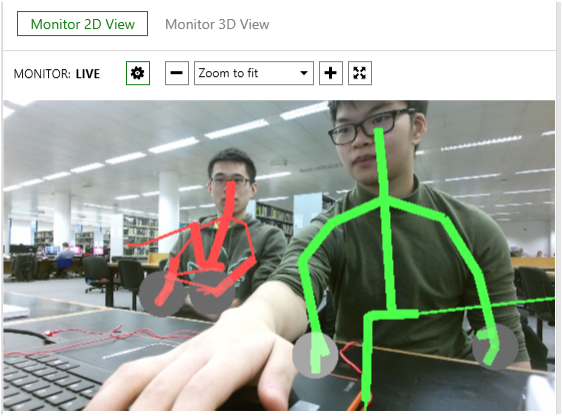


Figure 13: Colour Camera (with body tracking)

The V2 is definitely a great alternative to some of the more expensive depth sensors on the market. As highlighted in the paper "Low-cost commodity depth sensor comparison and accuracy analysis" by Timo Breuer, Christoph Bodensteiner, Michael Arens. Some of the other sensors that out perform it are upwards of 3000 while the V2 costs just around 150. This means that it will be accessible to a much larger audience, something definitely needed if an Augmented Reality game is to be successful.

Limitations and pitfalls

The cameras of the V2 have a set range of values that it works well within. This is reported by Microsoft to be from 0.4m to 3m [4]. Anywhere beyond that seems to be too far to determine the depth of objects. There is also a notion of being too near to the device to! If we are working with the device, say, next to our laptop or in a space nearby, there may be complications with how the sensor performs, anything closer than 0.4m produces unknown behaviour with the depth sensor.

A pitfall I expect to encounter is when developing with the Kinect SDK. I have experience in C++ which is one of the languages that can be used to write Kinect applications. However, it seems a lot of the colour camera tutorials are written in C# which will could cause problems for me if I had to pick up the language or translate it into the C++ equivalents. However, this should only be a small pitfall and cause minor hindrance to the progression of the project.

[4] https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges

4 Implementation

In this section I will briefly outline the main steps I took to in the creation of this project to tackle various problems.

4.1 Assumptions and Restrictions of the User

At the beginning of this project I had many ideas and aspirations on what the final product could do. However, I quickly realised that most of these ambitions were unattainable in the time frame and at my level of expertise. As a result, some restrictions have been imposed on the user when they use the program. In addition, I have designed my program with a set of assumptions in mind that simplify the problem from one that could merit several research projects if broken down and investigated in depth, into one of reduced difficulty but still contributes what it set out to do. In this section, I list these assumptions and restrictions.

4.1.1 Assumptions

- The user draws perfect connected contours or near perfect contours
- Light in the environment is stable and is not changing drastically over short time periods
- The user only draws contours
- The user draws reasonably sized landscape maps and contours
- The user doesn't draw an extremely deep tree of nested contours

I believe the assumptions I have developed this project with are reasonable, firstly, it would be very demanding if a user drew an incoherent landscape map and hope for the program to understand it. In addition, it will be unlikely a person will be drawing the landscape map in an environment with rapidly changing light. If the user draws other shapes or triangles, they will be interpreted as contours as this program only accommodates contours. It should also be reasonable to think that the user will not draw a miniscule landscape map as it becomes hard for both themselves and the camera. The case of a large map is fine as the restriction of seeing the ARuco marker keeps the scale within reason. The reason we do not want a deep tree is due to the amount of contour level traversal in the program. This should only become a problem if the user begins at the very edges of the paper and draws containing contours just one or two millimetres apart all the way until they fill the paper, which i a rather uninteresting landscape map anyway!

4.1.2 Restrictions

- The ARuco marker is visible in the Camera
- The pen colour must be noticeable on the paper colour
- The user must have a fixed camera
- The angle of the camera to the lanscape map can't be too small
- User must calibrate their camera themselves

In the ideal form of this project, there would be absolutely no restrictions on the user as this means that creativity is constrained to within the realms where these restrictions hold. However, as aformentioned, in this project having absolutely no restrictions became infeasible. The subject of markerless AR is being researched now curreny methods are rather cumbersome due to the real time requirement of AR. As a result, the user must have a marker on the surface they are drawing their landscape map. In addition, if it is hard for humans to see the contour lines, it is even harder for computers. As a result, the pen colour must be of and adequately different shade to that of the surface it is drawn onto. The user must also be able to calibrate their camera and make sure it is fixed in the scene such that it can see the landscpae map and not at too heavy of an angle otherwise the picture becomes way to difficult to recover.

4.2 DO Setting up the Camera and Environment

4.2.1 Camera Calibration and Setup

It is important in the use of the program that the camera being used to capture the scene is stationary. This is a fundamental restriction of the program. Althought it is possible to move the camera every so often, most of the time, it should be fixed to achieve the best results. The reason for this being that if the camera is on an unstable surface or is constantly experiencing some movement, the images captured will differ in pixels even though the landscape image itself has not changed. This will cause continuous recomputation of the landscape even though it does not need to be done. Extra computation imposes unecessary exra costs on the hardware of the user's machine and should be avoided. To user is permitted to move the landscape image in what direction they want, so long as it conforms the to restrictions outlines in section 4.1.

The camera used for this project was a webcam from Logitech, model C930e. The webcam can be seen in Figure 14. The camera has 1080p HD recording with scalable video coding. It records at 30fps and has a wide angle view with autofocus. It was very easy to

Figure 14: Webcam used in this project: Logitech C930e

mount this webcam and start using. The device is pretty high end, at the time of writing it is valued at $129USD which equates to around 85. Obviously this is not something everyone can instantly get their hands on due to its cost, however, the program will still function with much cheaper cameras, even in built webcams on laptops will work too (however, angling these to face the landscape map while having the screen face the user is probably impossible). However, this does prove that the webcam hardware is not a barrier to entry to start using this program to create landscapes.
To correct the issue of distortion within the web camera, I had to first calibrate it in order to figure out its intrinsic parameters. This was quite easily done by printing out a 7 by 5 square chessboard and running OpenCV's provided calibration program. The calibration is a program included within the OpenCV sample files and was run with the command
./cpp-example-calibration 0 -w 6 -h 4 -s 0.025 -o ../cam_0_params.yml -op -oe
where the various inputs are stating the nature of the board being used for calibration. 6 and 4 represent the number of internal corner points along each direction ($lengthofside-1$). The calibration process was very easy, just running the program while moving the chessboard in front of the camera in question, see Figure 15 will create a YAML file containing the intrinisic camera data and information about its distortion. The YAML file can then be passed into the program to help account for the the distortion and adjust the resulting displayed camera feed.

4.2.2 Choosing C++

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off. Bjarne Stroustrup

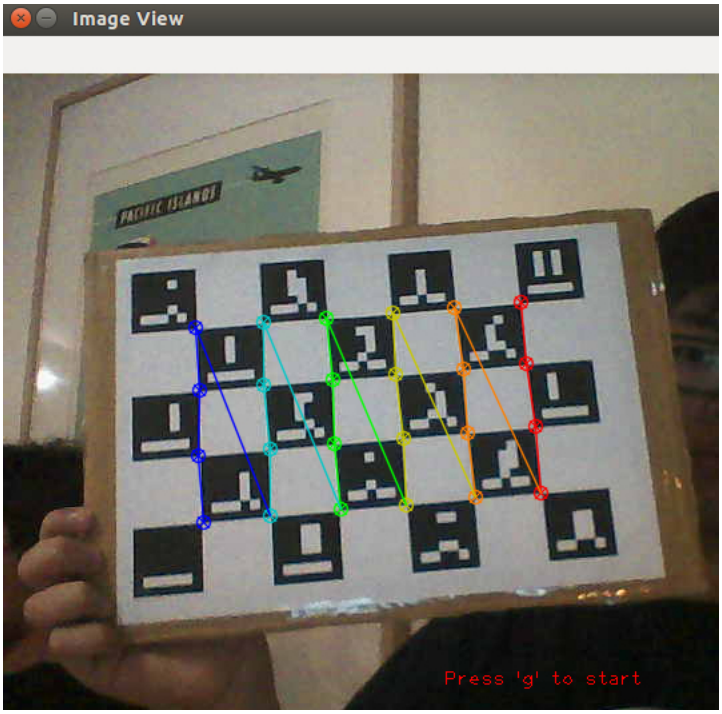


Figure 15: Calibrating the Camera

Regardless of this, C++ was the language of choice for this project. As a project that was concerned with Augmented Reality, it meant that I was definitely going to be messing around with Computer Vision and using Computer graphics to identify marker points and then render objects over them. In addition, it is a project that will be aimed at users of all kinds of ages, anyone requiring a creative environment should be able to use the end product. Augmented Reality is defined to have real time response and also requires custom 3D object generation and so will need to be very fast. Generally, this will mean that I aim to use a language that can be immedaitely converted into native machine code not something like Java that needs to be converted into Java byte code. In addition, this project utilised the Computer Vision library *OpenCV* and the Computer Graphics Library *OpenGL*. OpenCV is written C/C++ and has full support in C, C++, Java and Python, other language wrappers are avialable such as C#, Perl and Ruby. OpenGL supports various languages, however since we are working on a real time program, we want a language that compiles directly into machine code so that it is quick! As a result, the overlapping language was C++ and the language that was chosen to proceed with this project in. C was the other alternative, although due to personal experience and Bjarne's very relatable quote above, C++ was the language of choice. Even though there were various annoying points along the way in terms of debugging, I consider C++ the best, and probably only, choice of language I had.

4.2.3 OpenCV

OpenCV is the "Open Computer Vision" library, it is open source and was developed by Intel in 1998. From the year 2000, it has been under the BSD license, meaning it is a *very* popular library to use. In addition, it is probably the largest and most extensivel used computer vision library. OpenCV has a strong focus on real time applications, such as in video and image processing. In addition, some other areas that OpenCV is used in include Face recognition, Robotics, Motion Tracking, Augmented Reality and Interactions between Humans and Computers. In addition, OpenCV also includes some statistical machine learning tools, aiding with Decision Trees, KNN algorithsms, Naive Bayesian Classifiers amongst others. I shall not be utilising these in this project, however. OpenCV has various modules focused on different areas of Computer vision and offer numerous different transformations and operations on images and video such as segmentation, edge detection, object tracking and various others. Kim Yu Doo's paper[**?**] has a very nice summary of these moudles that OpenCV offers. One of these is the recent GPU module which allows computation to be done on the GPU to accelearate computation, this is highlighted in K. Pulli's (senior Director at NVIDIA research) article *Real-Time Computer Vision with OpenCV*[**?**].

There were numerous other Computer Vision libraries available, such as the Cambridge Video Dynamics library (CVD) and CCV, though due to the extensive documentation and

usage of OpenCV along with its massive user community, it was obvious to choose it over these. Using the OpenCV library has been very easy and seamless, with plenty of online documentation and guidelines. There are also numerous samples that come along with the library that helped me get to grips with using it for my project. The version used throughout was opencv 2.4.11.

4.2.4 DO OpenGL

OpenGL, "Open Graphics Library" was introduced when I was born, in 1992 and developed by Silicon Graphics and is supported by NVIDIA. It is the most widely adopted Computer Graphics Library. Akin to OpenCV, it has a massive community and a lot of online help, along with a Wiki page to support users. OpenGL interacts with GPUs to create computer graphics
.....MORE(Sort out opengl version)

4.2.5 DO ArUco

ArUco is a C++ Augmented Reality library that offers wrappers and methods based around OpenCV (versions 2.1 and above) that simplify the procedure of identifying fiducial markers in an image. The library is under the BSD license, created by the University of Cordoba in Spain. Release in 2014, this library is now ar version 1.2.5.

ArUco has been used in a multitude of projects already with success and has helped me immensely in the creation of this project. .....MORE

4.3 Capturing the Landscape map

Now that the environment has been set up for the user, we begin the first part of the Augmented Reality pipeline which is to conduct Computer Vision algorithms of the scene the camera is capturing. This corresponds to reading in the video feed of the landscape map that the user has drawn.

4.3.1 Drawing the map

During the implementation of the project, to initially start working with contour detection and landscape maps. I asked 2 friends along with myself to draw potential landscape maps. I used these to test the contour detection, initially just upon a static image basis, rather than using video feed. The landscape maps were drawn on typical pieces of white paper with black biro pen. The reason for using black biro pen is that it is the most common utensil for the average person to use to draw or write things. Other utensils considered were pencils and black marker pens. Pencils will allow erasing parts of the scene, however, there

will be gradient changes due to smudges and the effects of rubbing out pencil drawings. I decided to leave this scenario for a point later in the project or in an extension. Black marker pens are much thicker and thus easier to detect by vision techniques, however, I did not want to limit users to just black markers as they are not as commonplace as black biros.

The images were drawn in specific ways, they can be viewed in Figure 16.

Figure 16a has a couple of contours, closed, along with a few other shapes. The contours have some small artifacts(some bold lines and additional lines on the contour) which may happen in a typical drawing due to human error. The lines were drawn with intent, and although the other shapes are outside of the project's scope to identify as somethinig other than contours, this picture was used to illustrate a "typical landscape map".

Figure 16b was drawn by a friend who thought that labelling the contours with numbers to illustrate height would be a good idea. Unfortunately, this is outside the scope of the project, we are following the OS Map specification where each contour will dictate an uniform increase in height. By doing so, the friend has created a bunch of open contours which were used in this project to test the ability to close contours with small gaps. The ability to recognise numbers through template matching to manually set the heights may be a possible extension to this project but not something I will look in to.

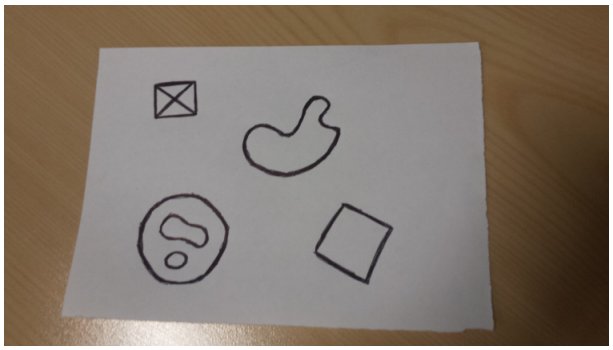
Figure 16c shows lines drawn in thickly, they are also all connected. This represents a landscape map where the user may have used a black marker or just thickly sketched out their design, making contour edges rough. This was made to mimic some noise as people may accidentally go over drawn lines and add small artifacts off the side of contours. This problem was not very large, however, due to the ability to use Morphological Closing as well as smoothing of the image with the Canny Edge Detector.

Figure 16d tries to mimick a bad situation where the user has drawn contours with large gaps, artifacts and odd other shapes on the landscape map. There are also some scribbles in the top right corner, placed as a test to see how far I could go with removing the unwanted scribbles. This drawing was more of a "lower bound" performance measure, to see how limited the program would behave and could be done to remove human error and what kind of restrictions and prerequisites a landscape map should have to be cause a decent Augemented Reality rendering.

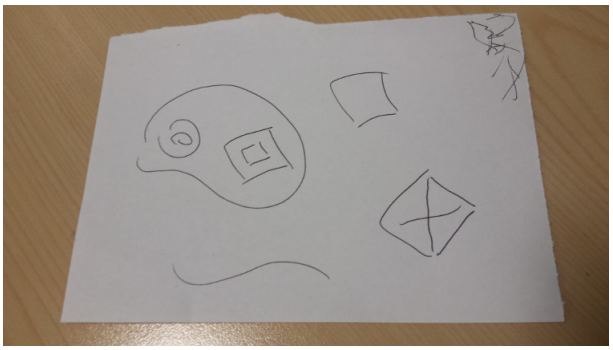
Figure 16e is a very simple image. Two contours, one inside the other, both closed. This is the basic iage that could be used to represent a "perfect" landscape map and is used to

(a) A variety of contours and other shapes.

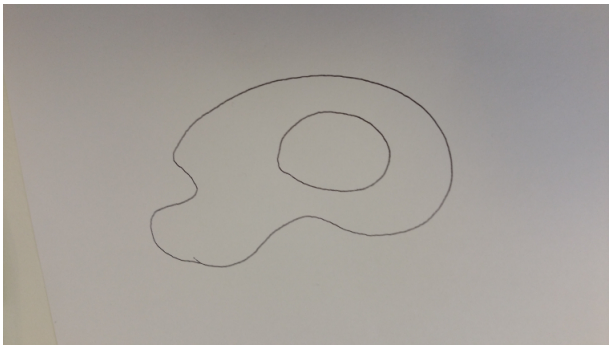
(b) All contours open, no artifacts.



(c) Shapes and contours drawn in thickly, all closed.



(d) Roughly drawn contours and pictures, som artifacts in the corner.



(e) Very simple contours drawn in; both closed.

Figure 16: Original landscape maps

test how well the program would work in the optimal conditions, if this isn't satisfactory, we cannot expect any of the other drawings nor the user's drawings to perform well at all.

Working on static images allowed me to mimic operating on frames taken from a webcam. As a result, operating in this way allowed me to easily and quickly make adjustments to threshold parameters and other operations on the images rather than having to extract from a camera each time and manually look at each result generated. Instead, it was much easier to load the image and run several bash scripts which output all the image results to files I could quickly view. I was very quickly able to attain empirical value for some of the thresholds and parameters in the function calls that OpenCV offers, these shall be presented in the next section.

4.3.2 Detecting Contours

Detecting contours is very simple when using OpenCV. A single function call will perform Canny Edge detection, as outlined in the background of this report, on the provided image. The function also performs double thresholding, so takes two threshold values that I had determined empirically, along with an aperture size for the Sobel operator and a choice of wether to use the L1 or L2 norm when determining gradient magnitudes.

The Canny function returns a list of contours and these are represented as a list of cartesian co-ordinates which dictate which pixels are considered part of an identified contour. The method call in C++ is shown below.

```
void Canny(img,contourImg,t1,t2,SAS,GMO)
```

1. `img`:
   Input image where we are looking for contours within.

2. `contourImg`:
   Is the resulting contour image produced by the algorithm.

3. `t1`:
   Lower Threshold, part of Hysteresis as mentioned in Section 3.3.4. Contours with a strength higher than this value are considered "weak contours".

4. `t2`:
   Upper Threshold, second part of Hysteresis, mentioned in Section 3.3.4 where contours of higher strength than this value are considered "strong contours".

5. `SAS`:
   Sobel operator Aperture Size. The Sobel operator is used to detect edges using gradient change and its size can be set here. The operator is a square of length set here.

6. `GMO`:
   Gradient Magnitude Option, specifies how to compute norms of image gradient magnitude.

The resulting `contourImg` is what is then passed onward through the pipeline for more processing by Computer Vision methods.

4.3.3 Generating the Hierarchy Tree

In openCV, there is a function that will take a contour image and from it, figure out the points of the image that are on contours. The function is called as below:

```
void findContours(imgIn,contours,hierarchy,mode,method,p)
```

The variables `contours` and `hierarchy` are variables into which the outputs of the function are placed. The `contours` variable is a vector of contours, which themselves are represented as vectors of pixel points. The `hierarchy` variable is a list of all the contours and their relation to the other contours (based on what is passed to the `mode` variable in the method). I used `CV_RETR_TREE` to return a list which describes a tree like hierarchy of the scene. Take for example the contour image (computer generated) by the landscape map in Figure 17. The returned hierarchy is seen below (has been formatted for easier reading).

```
//Contour N: [Next,Prev,FChild,Parent]
//Next = Next Contour on the same level as this contour (-1 if none)
//Prev = Previous Contour on the same level as this contour (-1 if none)
//FChild = The first child contour of this contour (a child is a contour con
//                  within another contour)
//Parent = the Parent contour of this one (-1 if none)

hierarchy = [
        Contour 0: [-1, -1, 1, -1]
        Contour 1: [-1, -1, 2, 0]
        Contour 2: [-1, -1, 3, 1]
        Contour 3: [-1, -1, 4, 2]
        Contour 4: [8, -1, 5, 3]
        Contour 5: [-1, -1, 6, 4]
        Contour 6: [-1, -1, 7, 5]
        Contour 7: [-1, -1, -1, 6]
        Contour 8: [-1, 4, 9, 3]
        Contour 9: [-1, -1, 10, 8]
        Contour 10: [-1, -1, 11, 9]
```

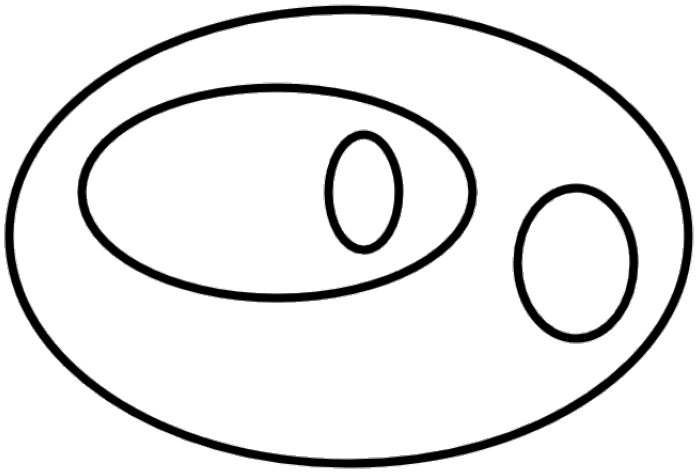


Figure 17: Computer generated landscape map

```
        Contour 11: [-1, -1, 12, 10]
        Contour 12: [-1, -1, 13, 11]
        Contour 13: [-1, -1, 14, 12]
        Contour 14: [-1, -1, 15, 13]
        Contour 15: [-1, -1, -1, 14]
]
```

To efficiently use this hierarchy, I actually create a Tree Object that encompasses the meaning of the hierarchy. The objects are included below.

```
class TreeNode {
public:
  //Methods and constructors removed
private:
  int level; //Level of the node in the tree
  TreeNode* parent;
  vector<TreeNode*>* children;
  TreeNode* next;
  TreeNode* prev;
  int nodeID; //ID of the node, as assigned by method findContours
};

class Tree {
```

```
public:
  //Methods and contructors removed
private:
  unordered_map<int,TreeNode*>* allNodes; //all nodes in this tree
  void insertNode(int,TreeNode*);
};
```

I keep this hierarchy as a global object in the program that is used whenever there is anything concerning contour manipulation as well as when creating the 3D scene which is explained later within this report.

4.3.4 DO Joining Contours

Joining contours is a key part of this project. As can be seen in the section 4.3.3, the landscape map in Figure 17 produced 15 detected contours after being passed to some OpenCV methods. Looking at the landscape map, it is extremely clear to us humans that there are ony 4 contours present. This means that Canny detection has probably split up each contour into 3 or 4 different contours that are very close but have gaps in between preventing them from being considered as one full contour. The returned hierarchy is thus not helpful to us at all as the realtionships between contours is inacccurate. Our primary goal now becomes to join contours before analysing any hierarchical data.

The first step to join contours is to perform morphological closing on the image. In Section 3.3.9, morphological closing was described as dilation followed by erosion. I do it in the opposite way this time because we want the *darker* areas (pen!) to expand rather than the typical brighter areas to dilate.

```
void morph_Closing(Mat* image, float elementSize, Mat* output){
  Mat structuringElement = getStructuringElement(MORPH_ELLIPSE,
        Size(elementSize,elementSize),
        Point(ceil(elementSize/2.0f), ceil(elementSize/2.0f)));
  Mat erodedImage(image->rows, image->cols,DataType<float>::type);

  erode(image, erodedImage, element);
  dilate(erodedImage,output,element);
}
```

The next step is realising that sometimes, a single contour is split into two but since they actually are both from the same contour, their end and start points are very close. With this in mind, I implemented a `naiveContourJoin` function. The function call is shown below and the corresponding algorithm shown in Algorithm 1.

```
vector<Point>* naiveContourJoin (vector<vector<Point> >*contourList,
                vector<vector<Point> >* joinedList)
```

Algorithm 1: Joining contours based on proximity of start and end points.

```
begin
    contourList is a vector of all contours, each contour is a vector of
    points which make up the contour.
    joinedList is the vector of returned contours.

    currentContour ⟵ first contour in contours
    for contour in contours[1..length(contours−1]) do
        if adjacent(currentContour,contour) then
            join(currentContour,contour)
        else
            add currentContour to joinedList
            currentContour ⟵ contour
        add currentContour to joinedList
        update the Hiearchy Tree
```

The function `naiveContourJoin` works by traversing the list of contours and examining in turn whether two contoursare adjacent, due to the nature in which OpenCV identifies contours, we can safely consider contours adjacent to each other in the list to check their adjacency. Adjacency is determined by whether the start or end of one contour is within a certain area of the strat or end of another contour. If it is, the contours are joined, if not, we move on down the list.
.....MORE(other methods)

4.4 Generating the 3D scene

After the initial Computer Vision aspect of the program has been completed, we can move onto the generation of the Augmented Reality scene that is overlayed onto the camera feed so that the user is able to view the landscape that they have drawn. From here, we have entered the Computer Graphics portion of this Augmented Reality program pipeline.

4.4.1 DO View Point alignment

4.4.2 DO From Hierarchy Tree to Height Map

Taking the resulting hierarchy from the Computer Vision stage of this program, we can then convert it into a height map which represents the state of the landscape. The height map is then used later on in the pipeline to help generate the landscape.

Creating the hierarchy tree allowed me to gracefully attach tree depths to each contour, encapsulated within a `TreeNode`. Thus, all that needs to be done is identify the contour that each pixel resides in and render its height as that equivalent to the depth of the `TreeNode` in the tree. However, this will create a massive "step" landscape which means that where there is a contour boundary, the pixels will suddenly jump from one height to another. This looks very unrealistic and not aesthetically pleasing. Since anything concerning computer graphics relies on looking good, it was a given to smoothen out the landscape heights. This was done with Euclidean distance transforms in mind, where we increase the height of the pixel by a percentage of how far the pixel is to the next height level. In essence, I implement a basic interpolation function that assigns a height as described in the pseudocode in Algorithm 2.

Algorithm 2: Assigning heights to the pixels in the heightmap.

```
begin
    for h ∈ imageHeight do
        for w ∈ imageWidth do
            p ⟵ pixel(h, w)
            c1 ⟵ getContainingContour(p)
            c2 ⟵ getClosestContour(p, c1)
            baseHeight ⟵ getLevel(c1)
            d2parent ⟵ getDistanceFromContour(p, c1)
            d2next ⟵ getDistanceFromContour(p, c2)
            extraHeight ⟵ d2parent/(d2parent + d2next)
            HeightMap[h][w] ⟵ baseHeight + extraHeight
```

However this alrgorithm depends on the function `getContainingContour` which, though provided in OpenCV as `pointPolygonTest`, using the function does not provide results as expected. The `findContours` function, mentioned in **??** returns a contour list as well as a hierarchy. Even though the hierarhcy can be established, sometimes `pointPolygonTest` doesn't recognise the contours as fully closed and so will not return as expected. As a result, I offer an alternative algorithm, seen in Algorithm 3

.....DO BELOW

Algorithm 3: Corrected height assignment into the heightmap.

```
begin
    for h ∈ imageHeight do
        for w ∈ imageWidth do
            p ⟵ pixel(h, w)
            c1 ⟵ getContainingContour(p)
            c2 ⟵ getClosestContour(p, c1)
            baseHeight ⟵ getLevel(c1)
            d2parent ⟵ getDistanceFromContour(p, c1)
            d2next ⟵ getDistanceFromContour(p, c2)
            extraHeight ⟵ d2parent/(d2parent + d2next)
            HeightMap[h][w] ⟵ baseHeight + extraHeight
```

4.4.3 DO Landscape Generation

Generating the landscape is pretty simple. After having create the height map in the previous step, all that remains is a simple 1-to-1 map of each entry in the 2D array to a pixel point on the screen. I made use of the openGL primitive, Quads, to generate the landscape. The code snippet below shows how I take use the height map to assign .....MORE

4.5 Detecting change in the scene

To provide the user inputless functionality that is the focus of this project, there has to be some way to indentify when it is time to redraw the scene. In this section I describe how I determine change within the landscape map and whether it is significant enough to represent a user change instead of a small change in environment.

4.5.1 Generating the Base image

The Base Image is the current landscape map that is used for generating the Augmented Reality 3D landscape. This project was focused on providing users with an dynamically updating environment to tackle the problem that some similar applications out there have, which is having to tell the program when they want the scene rebuilt. For a creative individual, quickly and seamlessly being able to make changes to their creation is paramount and a key motivation for this project. Rendering the landscape over and over again based

on each frame is inefficient, and not really needed,thus I have implemented an algorithm that only requires re-render of a scene if there has been significant enough change, i.e. the addition of a new contour.

Right now, as the program starts, the first frame captured is considered to be the Base Frame. The Base Frame is held in memory and from there, constant comparisons to the subsequent frames are carried out to determine whether significant change has been taken place. After a threshold point, the Base can be considered to have changed and a new Base Frame will be stored.

4.5.2 Introducing change

Change in the landscape map occurs in two ways, they may simultaenously happen. The first way is by physically editing the landscape map. A user can add and remove contours by drawing them in or erasing them from the scene. Removal of contours, however, will probably cause detection of the other type of change which is positional change. When a user erases a contour, they will likely shift the paper, when the user is not erasing a contour, they may also rotate or move the paper around the camera view.

With each of these types of change, the scene has to be redrawn. Change can also be introduced in other ways, some include obsuring of the scene or the fiducial marker, drastic changes in illumination, objects moving into the scene etc. However, with regards to Section 4.1, I assume that such unlikely and controllable factors by the user will not occur and thus not cause problems in change detection.

However, I had to consider the event of small changes. This means slight changes in the scene, i.e. say the user accidentally bumped the table on which their camera sat, this causes a small misalignment in the camera. Another scenario is that the user accidentally brushes the landscape map with their hand and ever-so-slightly moves it from it's original position. In cases similar to these, the change is so insignifcant that it hardly demands a re-render of the scene and thus should not. Thus, to counter this, I introduce various change thresholds which any change in the scene must surpass to count as adequate change for a redraw.
.....MORE (DO THE PICTURE TESTS AND THRESHOLDS)

4.5.3 Determining Stabilisation

The key part of creating the dynamic, self-updating environment is determining when it is time to update. The reason for this is that there should be an interval between a scene update that is not too frequent such that it imposes a large, unecessary amount of computation on the system. In addition, we do not want it so infrequent that even after the

user draws a new contour, there is still time before the recomputation occurs as this will take away from the real-time component whichh is so fundamental to Augmented Reality. A better way of choosing the time to update the 3D scene is to try determine *when* there has been a change in the landscape map.

As mentioned earlier, the way I have chosen to do this is through a comparison of captured frames to an identified base frame. When there has been enough difference captured, we can say that this is a suitable time to update the 3D scene and the base frame. However, the key here is to decide when we should actually update the base frame. There can be a massive change in the scene which passes the threshold, for example, the user moving their hand into the scene to draw in a new contour, every frame will be massively different to the next due to the amount of movement going on in the scene. Thus, the problem now becomes when can we say that the user has finished making edits to their masterpiece?

The solution implemented was to look for stability. After substantial change has occured, if the subsequent frames show little change, then it is likely the user has finished their changes to the scene. Thus, after this time has elapsed, we consider the first frame in the examined sequence the new base frame and this will trigger a redraw of the 3D scene. There are two ways in which change is identified in a base frame. These are between the landscape map or a change in the position and/or orientation of the ArUco marker.

4.5.4 Generating a Difference Image

Generating a Difference Image is rather simple. In the program, there is a global variable, `BASEFRAME`, which is the current stored base frame. Obtaining the base frame is explained in section 4.5.1. The code for this is shown below and requires 3 simple calls to OpenCV functions. First, the base frame and the current frame are converted into grayscale for ease of comparison. An absolute difference is calculated between the two images, pixels with a difference higher than `CHANGE_THRESHOLD` are marked as they have changed significantly (as described by the threshold).

```
/*** Elsewhere in the code ***/
//Converts into grayscale images
cvtColor(BASEFRAME,gBASEFRAME, CV_BGR2GRAY);
cvtColor(thisFrame,gthisFrame, CV_BGR2GRAY);
/*****************************/

//Create an image to store the Difference Image
Mat* diffFrame = new Mat(BASEFRAME.rows,
                         BASEFRAME.cols,
```

```
                                DataType<float>::type) ;

    //Get the absolute difference between the base frame (note:
    //these are the grayscale versions) and the
    //most recent frame, store this in the Difference Image.
    absdiff(gBASEFRAME, gthisFrame, *diffFrame);

    //Makes a binary image, pixels which have seen a change
    //higher than CHANGE_THRESHOLD are set to 1, the others are 0.
    //CHANGE_THRESHOLD is chosen empirically. diffFrame is the final
    //binary image and can be used later.
    threshold(*diffFrame, *diffFrame, CHANGE_THRESHOLD,
                    MAX_COLOUR_VAL, THRESH_BINARY);
```

4.5.5 DO Tracking the Fiducial Marker

4.5.6 Detecting Stabilisation

After figuring out how to detect change, all that remains is to use those as ways to indicate stabilisation. Due to the sparse spread of code for the project, I have instead included the pseudocode algorithm I used to detect stabilisation in the image, seen in Algorithm 4.
The logic behind the algorithm is as follows, from the base frame each subsequent frame has the potential to become the new base frame, should there be enough change between the two. Every time there is a significant difference between the last potential base frame and the next frame, the potential base fram egets updated. If there are frames after the potential base frame has been set that show change *underneath* the threshold then the stabilisation counter, `StabilisedCounter`, will be incremented. When `Stabilised Counter` has surpassed a set threshold value, it indicates that there is high posibility that the scene has settled from the last potential base frame update. What this transates into is that it is likel the user has now moved their hand out of shot of the paper and finisehd their change to the scene. When stability has been achieved, a re-render of the scene and recaluclation of the hierarchy tree and height maps need to be done. Other stabilisation variables are reset for next time a change might occur.

4.6 DO Restructure and regenerating the 3D scene

After substantial difference has been identified by the program, there is now need to re-structure and re-render the scene.

Algorithm 4: Detecting Stabilisation

```
BaseFrame is an image,
PotentialNewFrame starts off as equal to BaseFrame,
PotentialChange is a global flag,
StabilisedCounter is a global counter of frames.
begin
    while Camera is open do
        thisFrame ⟵ Next frame from Camera
        differenceImg ⟵ getDifferenceImage(BaseFrame, thisFrame)
        changedPixels ⟵ countNonZero(differenceImg)
        if changedPixels > FRAME_change_threshold then
            PotentialNewFram ⟵ thisFrame
            PotentialChange ⟵ true
        else if PotentialChange is false then
            StabilisedCounter++
            if StabilisedCounter >change_requirement then
                StabilisedCounter ⟵ 0
                BaseFrame ⟵ PotentialNewFrame
                PotentialChange ⟵ false
                Redraw the scene
```

5 Evaluation

In this section, I assess the outcomes of this project, trying to give numerical representation to its overall successes and failures.

5.1 DO Test Measures

In Table **??** below, I have compiled the various test measures I look into in an attempt to gain a representation of how well this project has turned out. For each, there is an associated unit of measurement in which I quantify the trait being assessed. The expected results correspond to what I originally considered the minimum result to be considered a success in that trait.

Trait to measure	Unit of Measurement	Reason
Contour Detection	Number of contours	A successful program will be able to distinguish c
Landscape Generation Accuracy	By observation	Since the accuracy of the generation both depend
Program Responsiveness	Seconds	Being real-time is a key part of Augmented realit
Base Frame Stabilisation	By Observation	We want the program to re-render the scene only

5.2 DO Indicators of Success

5.3 DO User Testing

Being a creative, Augmented Reality environment for people to bring their landscape creations to life, to assess its worth, there has to be feedback from the users.
Below I have compiled some of the responses from the users who took part in trying out this program.

5.3.1 DO Responsiveness

5.3.2 DO Aesthetics

5.4 DO Results

5.5 DO Overall Evaluation

6 DO Extensions

Having implemented an environment for dynamic creative Augmented Reality, content, there are many obvious improvements that can be made to transform this project into a much larger and better application.

The original idea for this project was the concept of having an environment for individuals to create their own 3D, responsive landscapes that can react to changes in the real world scene and reflect them within the AR scene. Indeed this has been done to a certain extent, changes in the landscape drawing cause the AR scene to change as a result, without need for user interaction with the program. Currently this only works with contour to landscape conversion. This will be of little use to individuals who wish to create something more than just land. In this section shall highlight some of the more immediate extensions and then go on to explore other extensions which have been lingering in the background of my vision for what this project could have become.

6.1 Immediate Extensions

There are numerous extensions that can be added to this project due to the vast amounts of things that could be brought to life by Augmented Reality. These are the extensions I would have implemented had I more time to commit to this project.

6.1.1 Coloured Contours

Currently the program works by taking frames from the camera and then processing them with various operations, such as perform morphological closing and blurring by the Canny Edge detector. One additional thing which can be done is to allow and detect coloured contours. Right now, if three sets of contours drawn in different colours were presented to the program, they would all behave the same way and create a terrain with a green shade. If we were to introduce new colours, we could also introduce new terrain! For example, brown tinted colours could represent rock or mountainous landscape; green can be used to represent something like everything within is a cluster of trees i.e. a forest! Blue can signify the enclosed area is a lake or a body of water. The things different colours

can represent are endless and will add an extra amount of creativity (and enjoyment) to the application! One use for this is creating a landscape for a battlefield or any other kind of more detailed environments. These in turn can be used to protoype landscapes for games!

To implement this wouldn't require much extension to the program. Just a simple reading of the contour points' RGB values and from there making a distance measure which will allow classification of the considered pixels to the closest colour. From there, generation of the 3D scene will be a switch case between different terrains base on the result of the classification.

6.1.2 Other markers

A marker is something on the paper or creative environment that will be detected and understood by the program. Right now, the only markers we have at the user's disposal are contour lines which are converted into land with different elevation levels. While it is very easy to add special computer generated markers .....MORE..... it removes from the creative experience of the application as there is less interaction with pen and paper.

To have other markers which represent other objects would greatly enhance the creative power of the user. Some examples of these are converting crosses ("X") into trees, drawn stars into houses or buildings, we can even add in animals or people with more markers and have them traverse the scene. Other than this last point, the other marker ideas require little extra logic to implement. By having a lookup table of shapes used as markers and what they represent, we can easily perform a template matching algorithm. OpenCV offers a matching algorithm which compares two images. We can extract markers drawn onto the surface and compare them against stored markers by passing the stored marker over the image as some kind of mask for convolution. A difference measure is run across the two images (taking into account rotation and affine transformation). If this is smaller than a threshold then it can be said within some statiscal region, that the detected area was intended to be that marker. From there placing the object over the mid point (with regard to surface normal) will suffice.

6.1.3 Photorealistic lighting environment

Augmented Reality comes under the fields of Computer Graphics and Computer Vision, and as a result it means that for it to be good, it has to look good. The potential user of this program can range from professional land/environment planner all the way to a 3 year old child with a bunch of pens at their disposal. If it does not appealing, it will not appeal to the professional and the sustained interest of the 3 year old cannot be guaranteed by

sub-standard graphics. A very easy way to quickly improve a scene and make it realistic, as Augmented Reality should be, is to add illumination equivalent to the local scene.

Implementing this also shouldn't require too much extra logic and is something I wanted to be able to add into the project; unfortunately I had not enough time due to unexpected road blocks within other areas of the project. By using a light probe placed within view of the camera, global illumination can be achieved without much intereference to the scene. A single frame can be enough to get the lighting for the environment. In addition, this can change with time as the environment can be update every so often wither by frame count or program run time, where another frame will be taken and the environment recalculated in the background. As a result, when lighting conditions change over time, it will cause no problem to the user and require no extra interaction.

Another point to think about is shadowing and self occlusion which should cause shadows to form upon the landscape createaed by other parts of the landscape itself. This would be included once the global illumination has been captured.

6.2 Future Extensions

I shall now highlight some of the other ideas I had while going into this project although some were a bit too broad and not realted enough to the main goals and contributions of this project. These extensions are more of what can be done to grow this project into a fully functions piece of software that could potnetially be used commercially, or indeed lay groundwork for future work in Augmented Reality. There are also ideas here that could be implemented when the technology for its realisation becomes available.

6.2.1 The Leap Motion

Before coming into this project I had a great interest and desire to work with some of the newer technologies such as touchless controls, Virtual Reality head pieces and Augmented Reality. While investigating these, I ran into the Leap Motion, an infrared hand tracker which allows you to use your hands as an input device. I really loved the idea of using your own hands to interact with things in a generated environment as opposed to touchless control of a device. However, the leap motion still has a number of bugs and is not as seamless and hand tracking as one would like it to be. It is still one of the better and cheaper hand tracking devices out there, with advances in its development I think it could prove very useful to users of the AR program created from this project.

The basic idea for integrating this piece of hardware into the project is to, after creating the scene, the user is also able to *manipulate* the scene they had just created! This adds

another dimension onto the creative ability of the the application and really promotes the basic idea I have when conducting this project. Using your hands to create things!

On top of this basic idea, there were a variety of potential pieces of functionality that could be implemented with this new medium of interaction, some of which I shall list:

1. *Allow the user to scale the environment using their hands.*
   By registering a pinch action, if the user pinches edges of the generated scene, or by pinching the highest point and pulling outward/upward, the user can scale the generated scene to their desired size! Will require alignment of Leap Motion hand point coordinate space to the 3D scene coordinate space to work.

2. *Adding content by drag and drop.*
   This functionality is inspired by games much like *The Sims*. The user can choose to either add in content through the computer and literally drop it into the scene by moving their hands around. This could possibly be achieves by pinch and release as aforementioned. If the extension of adding in animals/people or other moving object that traverse the scene were to implemented, the user could also pick these up and move them around the scene at will. This will introduce more of a game aspect into the project and will definitely appeal to the younger set of users while adding functionality for those who could be using this for professional work.

3. *Introduce "God Mode"*
   Again inspired by some games and similar to the above point. Allow the user to view hands in the scene, as if you were playing the role of God. This is purely to enchance the program as more of a game instead of a creative environment. God mode would allow the user to make various changes to the environment, for example spreading the hand could represent rainfall on the area below the convex hull of the hand points! Another idea is if the user start pressing on the landscape, the landscape itself can change, e.g. decreasing in height, cause cracks in the scene etc. This is one of the more ambitious extensions.

6.2.2 No Calibration Marker

The bigger dream for this project was to enable users to be creative wherever they go with their computer or laptop. The idea was to have a program that imposes as low an amount of restrictions on the user as possible. One of the main restrictions is the calibration marker. In its current state, for the user to begin their creative landscape maps, they have to be in possession of paper marked with an ARuco marker. In addition, this marker has to be of a specified value and size.

In order to further improve the application, it would be great to remove the need for a calibration marker. This means that the user does not even have to carry around paper, they can grab any scrap piece that they find around should an idea spring to mind. There have been a few papers and investigations in Augmented Reality without fiducial markers. Ferrari gives a good introduction into some methods for a real time AR system without explicit markers in the paper "Markerless Augmented Reality with a Real-time Affine Region Tracker" [**?**]. In here, they rightly state that most AR researchers *often have to take refuge to putting markers or landmarks into the scene* due to the real time requirements of AR, as is no different in this project. There are also many applications that already work without the need for these markers, though many use some form of implicit marker rather than an explicit one. An example of this is the Augmented Reality dressing, where a user stands in front of the camera, implicit markers like hands, head, shoulders etc. are used to gain an understanding of the scene in front of the camera.

In order for this to work for my project however, there is need to somehow convert a part of the users' landscape map into a base for calibration and alignment. The mobile app *LandscapAR*, mentioned in Section **??**, uses the paper corners as basis; in my opinion that restricts the user to a certain environment (dark background, all four corners in camera, etc.) and means a lot more computation if tilt of the camera or scene has to be considered. There will have to be a lot more investigation and testing of implicit methods in the field of AR before I think this extension could come to fruition.

6.3 Continued testing and experimentation

For this project to come together, there were numerous problems that had to be solved. Truthfully, some of these problems could have been entire projects themselves and have been studied a lot in their field of literature. Since there is not enough time to explore each of these problems in depth, some simple approaches have been taken to reach a suitable solution that will fulfil the needs of this project. The main areas which I think would benefit from more extensive testing are highlighted in this section.

6.3.1 DO Contour closing

Contour closing is a pretty large topic in the field of Computer Vision as it is the basis for segmentation. As a result, it plays `the` main role in many applications of computer vision. As a result of this there are already numerous papers out there which look into achieving accurate segmentation. Some of these require human input, such as creating atlases in medical imaging. Machine learning methods can also require some forms of human input, mainly

in reinforcement learning where the program may segment an image by finding contours and then the user correct the segmentation, causing updates to the program's segmentation algorithm. Over time, this would hopefully improve the performance of the program when identifying contours in roughly similar images.

In this project, contour closing has been dealt with through a naive approach. Firstly, morphological closing helps connect very cloase contours, however, there were still contours that were very close that weren't caught by this transform. The next approach was to join contours that have start and end points which neighboured the start/end point of another contour. .....MORE

6.3.2 DO Thresholding

Computer Vision is heavily dependent on aesthetics. A successful Computer Vision application is not one that performs the best, but the one which looks, visually, the best. As users, we greatly desire things that look good! If landscapes looked horribly choppy and like they were made from the previous decade, users will generally not be satisfied with it. This is heavily evident in the game and movie industry where there is constant desire to make games and movies more realistic and not computer generated. Removing the invisible cloak worn by generated images that practically say "I'm not real" is the goal of many companies in this field. This is even more so in Augmented Reality, where even in the name it containes the word "reality."

In this project, there were various points where we applied thresholds to images in an attempt to better segment and divide parts of the image. Some areas include contour detection, ...MORE

6.3.3 DO Stabilisation of background

To determine if there was any change in the landscape map, we implemented an algorithm which takes a base frame and the current frame and calculates a change measure. The base frame was considered the "Background" and any change was the "Foreground".
.....MORE

References

[1] 09ggr820. Canny edge detection. Indian Institute of Technology Delhi, Department of Computer Science and Engineeering, March 2009.

[2] Clemens Amon and Ferdinand Fuhrmann. Evaluation of the spatial resolution accuracy of the face tracking system for kinect for windows v1 and v2. volume 6th Congress. Alps-Adria Acoustics Assosiation, 2014.

[3] Ronald T. Azuma. A Survey Of Augmented Reality. Technical report, Hughes Research Laboratories, August 1997.

[4] D. Bassily, C. Georgoulas, J. Guettler, T. Linner, and T. Bock. Intuitive and adaptive robotic arm manipulation using the leap motion controller. In *ISR/Robotik 2014*, volume Proceedings of 41st International Symposium on Robotics, pages 1 – 7. VDE, June 2014.

[5] John Canny. A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Pami-8(6):679–698, November 1986.

[6] Lijun Ding and Ardeshir Goshtasby. On the canny edge detector. *Pattern Recognition*, 34:721–725, 2001.

[7] Bartholomus Rudak Frank Weichert, Daniel Bachmann and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. Technical report, Department of Computer Science VII, Technical University Dortmund, May 2013.

[8] Wendy E. Mackay. Augmented reality: linking real and virtual worlds: a new paradigm for interacting with computers. Technical report, Department of Computer Science, Universit de Paris-Sud.

[9] Raman Maini and Dr. Himanshu Aggarwal. Study and comparison of various image edge detection techniques. *International Journal of Image Processing (IJIP)*, 3(1):1–12.

[10] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: A systematic survey. Technical report, Rensselaer Polytechnic Institute.

[11] Rashmi, Muhesh Kumar, and Rohini Saxena. Algorithm and technique on various edge detection: A survey. *Signal and Image Processing: An International Journal*, 4(3):65–75, June 2013.

[12] S. Reed, O. Kreylos, S. Hsi, L. Kellogg, G. Schladow, M.B. Yikilmaz, H. Segale, J. Silverman, S. Yalowitz, and E. Sato. Shaping watersheds exhibit: An interactive, augmented reality sandbox for advancing earth science education. University of California, Davis and American Geophysical Union (AGU) Fall Meeting 2014, 2014.

[13] Paul L. Rosin. Thresholding for change detection. Technical report, Brunel University.

[14] G. T Shrivakshan and Dr. C Chandrasekar. A comparison of various edge detection techniques used in image processing. *IJCSI International Journal of Computer Science Issues*, 9(5):269–276, September 2012.