Active 3D Smartphone App

Frank Zhao, Australian National University
Supervised by Dr. Stephen Gould, Australian National University
frank.zhao@anu.edu.au

May 30, 2014

Abstract: A novel method of creating a side-by-side 3D stereogram from the foreground of a 2D image was implemented. To achieve this, the pipeline implemented two step interactive GrabCut image segmentation, followed by recreating the object in 3D with a depth map, and finally rendering two images side by side with slightly different perspectives in front of an inpainted background to create a stereo pair.

1 Introduction

More and more devices are incorporating camera hardware into their designs. As a result, there have been many mobile applications released for mobile platforms that perform image manipulation without the requiring the user to leave the device. The hardware specifications of common smartphones are becoming more powerful for less cost, and a significant portion of users own smartphones that possess similar computational ability comparable to low end desktop computers. The combination of these two facts suggests that incorporating more computationally significant image manipulation tasks into smartphone targeted applications is quickly becoming feasible.

An novel application of these features is to make viewing photos more interesting. While a photo may preserve a memory, they are simply static slices of time, that has essentially no interaction between the human and the image beyond viewing. Generating interactive images from static, planar images is an interesting possibility to enhance the photo viewing experience.

1.1 Side-by-side 3D

Side by side 3D is one of the simplest forms of stereoscopy. A pair of images, known as a *stereogram*, contains two different images of a subject from two perspectives [5]. The deviation in perspective is adjusted such that it is similar to the deviation due to binocular vision in the human eye. A major advantage of side by side 3D compared to other stereoscopy techniques is that it requires very little equipment to view. In fact, a common method of viewing the stereogram is known as *freeviewing* and simply involves the user focusing their vision at infinity, while looking through the pair of images. The two images appear to converge, creating an illusion of depth.

This technique can be implemented easily on a smartphone display. If the stereogram is displayed on a smartphone screen and viewed from approximately 5 centimetres away from the eyes, the two images appear to converge, providing the illusion of depth. However, since the average human eye has a minimum focal distance of approximately 15 centimetres, it is impossible to view the stereogram correctly. In order to assist with this focusing, stereoscopes with strong lenses (>20 dioptres) can be used to view the image pair. A stereoscope contains two lenses that scale and shift the position of the image horizontally such that the two images combine into one. Stereoscopes

have become popular with hobbyists due to their ease of manufacture. A simple stereoscope can be constructed using widely available telescope lenses and some cardboard as the frame.

1.2 Proposal

A novel proposal was made to enhance the user experience of viewing a photo. A three dimensional representation of the foreground could be created as a virtual object with depth. This virtual object could then be viewed from a different perspective, generating an interactive image. Combining this with the technique of side-by-side active 3D, two images can be generated by observing the object from slightly different angles, one for the left eye and one for the right eye. These images can be combined to form a stereo image pair, that when viewed through a cross eye viewer, creates the illusion of depth due to stereopsis in human vision.

2 Previous work

Constructing this system requires the union of several problems in computer vision. Most of the challenges are solved or partially solved problems. In order to separate the foreground object from the background, an image segmentation algorithm was needed (GrabCut). After the image is segmented, a large void is left in the background where the foreground existed previously. This then needs to be filled using an inpainting algorithm (Navier-Stokes). Finally the foreground object constructed in three dimensional space requires some kind of object completion to evolve from a two dimensional planar image. In this case, a depth function based on Euclidian distance was used to give each pixel a corresponding depth.

2.1 GrabCut

Based on an energy minimisation solution known as GraphCut, GrabCut improves the segmentation by using a Gaussian Mixture Model (GMM) and an iterative minimisation procedure [3]. Segmentation is simply captured by an alpha channel of the image. The mask contains opacity values $0 < a_{(i,j)} < 1$. For the foreground segmentation, the mask would contain $a_{(i,j)} > 0$ for pixels that are identified as the foreground, and $a_{(i,j)} = 0$ for pixels that have been identified as the background. As such, the goal of segmentation can be expressed technically as finding values of α using a model θ and the image data z.

A Gibbs energy function can be defined so that a good segmentation corresponds to a minimisation of this function [4]. The Gibbs energy has a form similar to a Lagrangian, and has the form

$$E(\alpha, \theta, z) = U(\alpha, \theta, z) + V(\alpha, z) \tag{1}$$

The first energy term U is a function that determines the fit of opacity to the data. Low values are awarded if the opacity α provides a good fit to the colour model.

$$U(\alpha, \theta, z) = \sum_n -log\ h(z_n; a_n) \tag{2}$$

The second energy term V is determined by the smoothness of the fit. This term is large for values of opacity that exhibit significant disparity between neighbouring pixels such as outlying pixels, and rewards smooth clusters of similar identification.

Minimising this energy model results in a good segmentation. The minimisation problem can be solved by applying a minimum cut algorithm for the problem

$$\alpha = arg\ min\ E(\alpha, \theta) \tag{3}$$

GrabCut makes two main improvements to the original GraphCut algorithm [3]. First, the monochrome image model used by GraphCut is replaced by a colour Gaussian Mixture Model. This results in more accurate segmentation for images with regions that have similar grayscale intensity between segments, but have high contrast in colour. Additionally, while GraphCut uses a trimap to divide a monochrome image into three regions (background, foreground, unknown), GrabCut can infer the foreground and unknown maps. This reduces the segmentation algorithm to only one user interaction step: specifying the background region.

2.2 Inpainting

The goal of inpainting is to restore missing portions of an image with data that is consistent with other areas of the image. The Gestalt Law of Continuation states that lines are perceived to continue in their established direction, even when the line is non-continuous. Inpainting algorithms work by iteratively continuing the structure and colour of surrounding regions into the missing area [2].

The Navier-Stokes equations are a set of non-linear partial differential equations, commonly used in physics describe fluid dynamics. The problem of inpainting lends itself to solutions that are found through the application of equations that describe fluid flow, since satisfactory inpainting results requires continuous colours and boundaries. Likewise, propagation of intensity in a smooth manner makes inpainted regions of an image appear more continuous. Incompressible Newtonian flows obey the following Navier-Stokes equations [1]

$$v_t + v \cdot v = -\nabla p + \nu \Delta v \tag{4}$$

$$\nabla \cdot v = 0 \tag{5}$$

where v is the velocity vector, p is pressure and ν represents viscosity. For two dimensional flow, a stream function can be introduced such that

$$\nabla^{\perp} \psi = v \tag{6}$$

In the image inpainting problem, an analogy can be drawn such that this stream function becomes the image intensity. Furthermore, the fluid velocity vector is directly analogous to the isophote direction, and vorticity becomes a smoothness parameter. Repeated applications of Navier-Stokes equations to an image will propagate intensities and contours in their established direction, resulting in an inpainted image. These are known as Navier-Stokes based inpainting algorithms.

3 Implementation

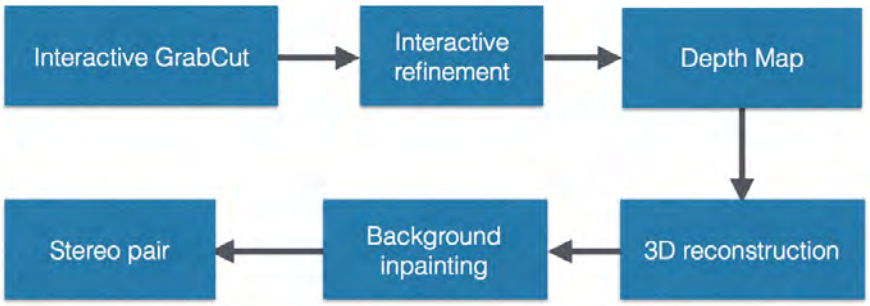

FIGURE 1. The implemented pipeline. The image is segmented, before a depth map is applied and the image is reconstructed in 3D space. Two different perspectives of the image are generated and the resulting stereo pair is rendered into two side by side OpenGL views.

An overview of the pipeline implemented is shown in Fig. 1.

3.1 Foreground segmentation

First the image is loaded and the user draws a rectangle around the foreground of the image (Fig. 2). A GrabCut algorithm then marks the region not enclosed by the rectangle as the background map, and creates two image masks, one for the foreground and one for the background. The image is then presented to the user with the background removed, showing the result of the foreground segmentation. The user can then proceed to perform refinement of the resulting masks by simply painting over the image. Two paintbrushes are available, one that marks regions as foreground, and one that marks regions as background.

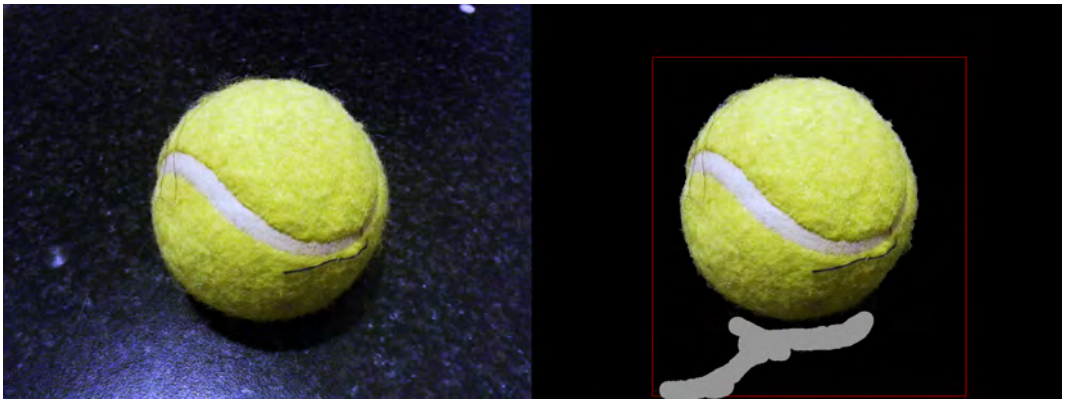

FIGURE 2. The user selects the foreground region of the image using a rectangle. GrabCut uses the inverse of this selection as a background map. The resulting foreground segmentation can then be refined by painting regions with a mouse (gray) to add or remove incorrectly identified regions.

Once the user finishes refining incorrect or neglected areas during segmentation, the GrabCut algorithm is run again, this time incorporating the user refinements in the initialisation mask.

3.2 Depth function

The foreground is then given a depth map based on a distance squared function, resulting in a paraboloid that achieves a maximum at the centre of the image. The depth of a pixel is given by a function based on Euclidian distance

$$D_{i,j} = i_{max} - s\sqrt{(i_{max}/2 - i)^2 + (j_{max}/2 - j)^2} \tag{7}$$

where $D_{i,j}$ is the depth of the pixel with some scaling factor. This approach assumes that the foreground object is always at the centre of the image.

3.3 3D reconstruction

Each pixel is then converted into a vertex in OpenGL that retains the original colour of the image pixel. The transformation of a pixel on the image with a two dimensional coordinate and a depth value into a three vector is given by the following equation

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = s\ R\ \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \right] \tag{8}$$

where (u, v, w) is the coordinate of the vertex in three space, and (x, y) is the coordinate of the pixel in the image with a depth z. A translation is applied to the initial vector to place the vertex in the right position in space, before a rotation matrix R is applied in order to rotate the image plane perpendicular to the ground plane.

The rotation matrix R is given by the three dimensional rotation matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{9}$$

where the rotation angle θ is dependent on the height coordinate of the pixel in the image γ and the angle of the image plane with respect to the vertical axis ϕ

$$\theta = -\cos^{-1}\left(\frac{\phi}{\gamma}\right) \tag{10}$$

Finally a scale factor s is needed to ensure that the reconstructed object is at the right size. In order to render the image back onto the image plane, we can invert this operation using the relation

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R^{-1}\frac{1}{s}\begin{pmatrix} u \\ v \\ w \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{11}$$

where the parameters are the same as in the initial transformation. Fig. 3 provides a visual representation of this process.

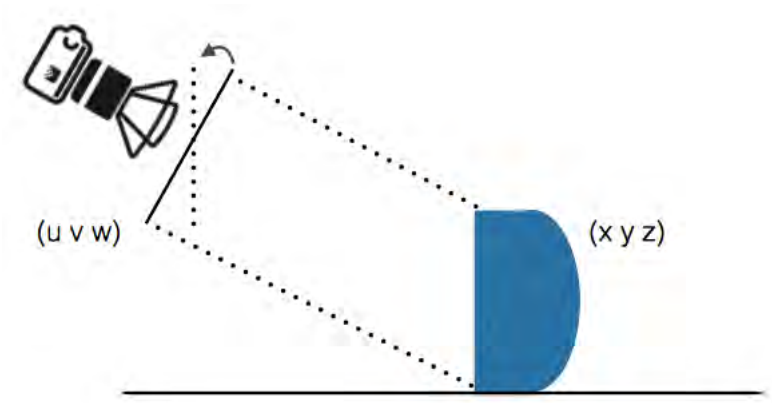

FIGURE 3. In order to construct the object in three space, points on the image plane (u, v, w) need to be rotated and translated into the correct position, before being scaled to the correct size. The resulting points (x, y, z) then form a three dimensional point cloud.

3.4 Rendering

After transforming each pixel into a point in three space, the vertices are then rendered as a triangular mesh (Fig. 4). To do this, two rows of the point cloud are considered. Each pair of points in the top row in addition with one point in the lower forms the upper triangle. Likewise, a lower triangle is formed with a single point in the top row and a pair of points in the lower row. The triangle is then rendered as a face in the overall mesh.

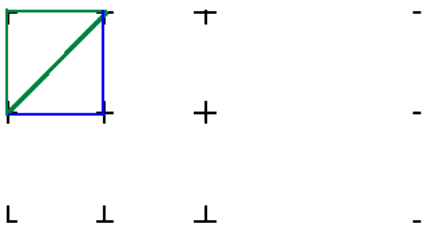

FIGURE 4. The points in the reconstructed point cloud is rendered as a triangular mesh. The upper triangle is shown in green, and the lower triangle is shown in blue. Together, the two triangular meshes form a square.

To construct the stereo images, the viewing camera is transformed laterally in either direction by a specified distance, and the view is rendered to the corresponding viewport. Finally, the missing background is then inpainted using a Navier-Stokes based algorithm in OpenCV, and rendered as a plane behind the foreground object.

4 Results

In most cases, the image segmentation produced satisfactory results (Fig. 5), although small refinements may be necessary (Fig. 6). Figure 7 shows the an example output of stereogram generation. Due to the simplistic function to assigning a depth value to each pixel, the pipeline only produces satisfactory results for objects with front facing surfaces that can be modelled as a paraboloid. While this means that images with spherical objects generally produce favourable results (Fig. 8, 10), images with more complex topologies produce significant distortion when rendered the perspective is changed (Fig. 9).

FIGURE 5. Left: the original image; **Right:** the segmented image, showing only the foreground.
The image segmentation works well for images that have a smooth, continuous background. For more complex background such as Fig. 6, additional refinement steps are needed. Additionally, since the GrabCut algorithm uses a Gaussian colour model, segmentation of foreground subjects that contain similar colours to the background is limitation of this implementation, as it results in missing foreground regions.

FIGURE 6. Foreground segmentation of images that contain complex backgrounds require manual refinement. In this case, some grass remains above the dog's head. The refinement process is shown in Fig. 2.

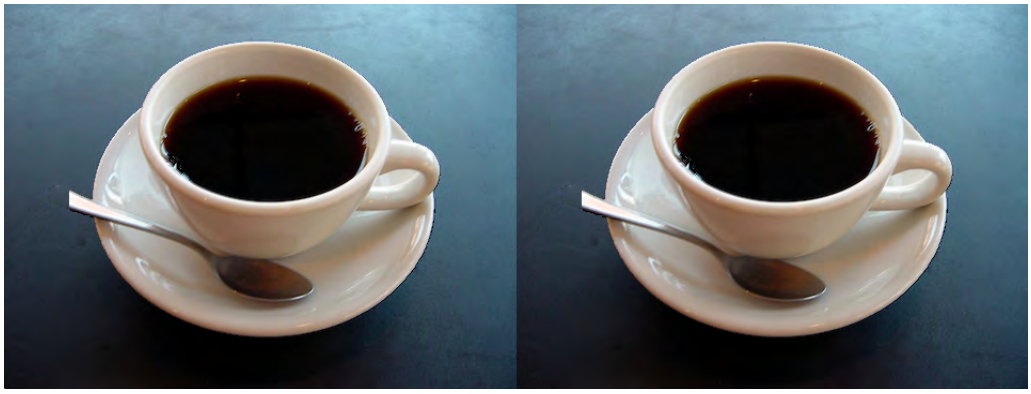

FIGURE 7. An example of a stereogram constructed by the program. The each viewport corresponds to an image rendered from a perspective that is shifted horizontally in the corresponding direction. The distance between the perspectives correspond to the natural distance between the viewer's eyes.

FIGURE 8. Left: original image; **Middle:** the reconstructed object with a small rotation in perspective; **Right:** a large rotation.
A demonstration of the depth function that is applied during reconstruction of the foreground object. The depth is a paraboloid that achieves maximum at the centre of the object. Rotating the viewing camera around the object, the depth function creates an accurate illusion for small rotations, but significant distortion is inhibited for large perspective changes.

FIGURE 9. Left: a horizontal perspective shift; **Right:** a vertical perspective shift.
For more complex objects with varying depth regions such as a coffee cup, the depth function is obviously an incorrect interpretation of the object. Assuming the foreground object bulges out from the centre is not an accurate representation of its true form. As a result, changes in perspective lead to distortion effects.

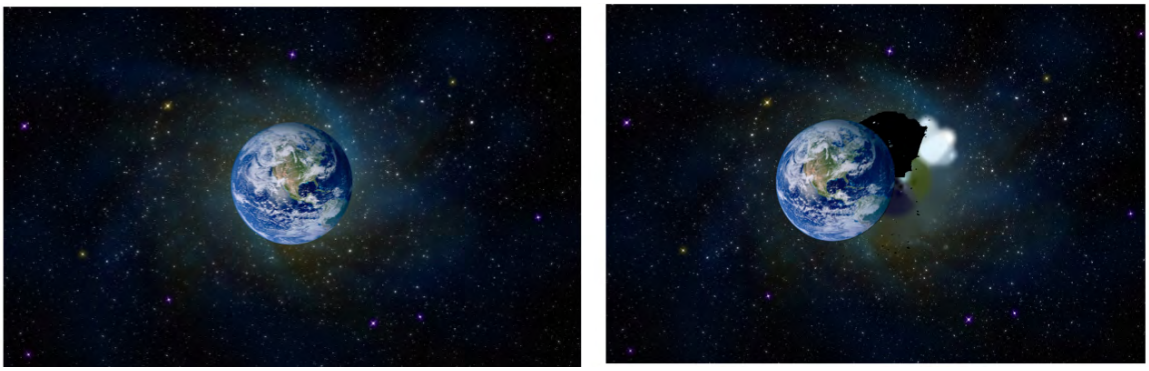

FIGURE 10. Left: original image; **Right:** reconstructed image.
The implemented depth function lends itself to work well with spherical objects. In this case, a globe is segmented and reconstructed. For limited changes in perspective, the depth function provides a reasonable illusion texture continuation into the rear of the object. Note that the black region above and to the right of the reconstructed image is an example of incorrect refinement after segmentation.

5 Discussion

Currently the major flaw in the pipeline is in the depth function. Assuming that the object surface can be represented as a paraboloid restricts good results to a limited class of images. This assumption results in distortion of images that do not exhibit a paraboloid surface (Fig. 9). Real world objects often have much more complex topographies, and this depth information is difficult to infer. Many applications that require depth information include hardware solutions such as infrared depth imaging in the recording stage of the image. One possible improvement to inferring the topography of an image is to incorporate edge detection into the image processing pipeline. Edge detection would allow the easy identification of contours, and under most lighting conditions, this provides a good inference as to regions of changing depth. A further user interaction step can be added to this, allowing the user to specify the depth of the contours.

For the program itself, the implementation is also very inefficient. mages are often represented as a matrix of pixels. Most photographs are stored in a 3-channel RGB format, resulting in a $3nm$ element matrix. Each element contains information about the value of channel colour corresponding to the index for the image. For a ten megapixel camera, this corresponds to approximately 9.5 megabytes of raw data. It is immediately obvious that this has a significant memory footprint on a mobile device. Additional required data such as the depth map and segmentation masks, the required memory for the pipeline is at least a significant multiple of the raw image data.

Similarly, matrix operations are computationally expensive. Most matrix operations require looking at each element in the matrix at least once, and common operations such as matrix multiplication have time complexities of $O(n^3)$. Currently, transforming the object into three space, and updating the projection of this transformation after camera rotation takes the longest to complete. The performance was found to be very slow on images of more than 0.5 megapixels or higher, and even images that were smaller required an amount of time to process that would not be feasible for an end user. Since cameras today have very high resolutions of around 8 megapixels, images will need to be resized before processing. Likewise, there is much optimisation to be done in order to create an application that can run on mobile hardware with limited computational resources. One suggestion for optimisation is to take advantage of the multicore and GPGPU capabilities of smartphones, since these features are becoming more widely available. It is highly likely that GPGPU implementations of matrix operations will significantly accelerate the rendering process.

Another limitation of the implementation is the assumption of key parameters. Although the distance of an object in reality can be inferred from information such as focal distance in the image metadata, this is not always available. Similarly, the camera height and image plane angle are all important parameters in the reconstruction that must be inferred. The camera height directly influences the height component of the translation vector, while the rotation of the image place directly determines values in the rotation matrix. In order to consider these parameters, scene understanding or horizon algorithms can be used. Although these methods can be considered, it is likely that this is an additional step of user interaction that will be required.

The stereogram generation also required additional parameters that are not present in the image. Perhaps the most critical of these for stereogram viewing is the perspective distance between the viewer's eyes. Since this value is different for individual parameters, the ability to adjust these settings is required. Ideally this can be adjusted in stereoscope hardware, but this feature is necessary to increase user comfort. Likewise, the large variety of screen resolutions and pixel densities of display devices needs to be considered, in order to create a device-tailored stereo pair.

6 Conclusions

Side by side stereo 3D provides a simple and cheap solution to creating more interesting images. Creating an effect where the foreground of an image pops out, lends itself to a simple application

of stereoscopic images. A pipeline was implemented to segment the foreground of an image and generate a stereogram from a reconstructed foreground point cloud. While the results were satisfactory for small binocular perspective distances, the pipeline is restricted to performing well on a small subclass of image subjects due to the simplistic depth function. The major challenges for future work include creating a depth function that is accurate for more foreground subjects, as well as optimising the pipeline such that satisfactory performance and efficiency can be observed on hardware with smartphone similar resource and computation limitations.

7 Final words

Many thanks to Dr. Stephen Gould for his advice and supervision. The code for this implementation has been made available on GitHub: `https://github.com/frankzhao/active3d`.

References

[1] Ryo Takei Wilson Au. Image inpainting with the navier-stokes equations. *APMA*, 2011. http://elynxsdk.free.fr/ext-docs/Inpainting/todo/inpainting%20with%20the%20Navier-Stokes%20Equations.pdf.

[2] Roman Stanchak. Image inpainting; navier stokes, fluid dynamics and image and video inpainting. http://mm.cse.wustl.edu/perceptionseminarresources/inpainting.

[3] Andrew Blake Carsten Rother, Vladimir Kolmogorov. âĂIJgrabcutâĂİ âĂŤ interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (SIGGRAPH)*, 2004. Microsoft Research Cambridge, UK.

[4] Xiaoqian Xu Justin F. Talbot. Implementing grabcut. 2006. http://research.justintalbot.org/papers/Grabcut.pdf.

[5] Marcus Hutter. Diy 3d virtual reality goggles. 2013. http://www.hutter1.net/puzzles/3dvrglass.htm.