Robust Semi-Automatic Depth Map Generation in Unconstrained Images and Video Sequences for 2D to Stereoscopic 3D Conversion

Raymond Phan, Student Member, IEEE, and Dimitrios Androutsos, Senior Member, IEEE

Abstract

We describe a system for robustly estimating synthetic depth maps in unconstrained images and videos, for semi-automatic conversion into stereoscopic 3D. Currently, this process is automatic or done manually by rotoscopers. Automatic is the least labor intensive, but makes user intervention or error correction difficult. Manual is the most accurate, but time consuming and costly. Noting the merits of both, a semi-automatic method blends them together, allowing for faster and accurate conversion. This requires user-defined strokes on the image, or over several keyframes for video, corresponding to a rough estimate of the depths. After, the rest of the depths are determined, creating depth maps to generate stereoscopic 3D content, with Depth Image Based Rendering to generate the artificial views. Depth map estimation can be considered as a multi-label segmentation problem: each class is a depth. For video, we allow the user to label only the first frame, and we propagate the strokes using computer vision techniques. We combine the merits of two well-respected segmentation algorithms: Graph Cuts and Random Walks. The diffusion from Random Walks, with the edge preserving of Graph Cuts should give good results. We generate good quality content, more suitable for perception, compared to a similar framework.

Index Terms

2D to 3D image conversion, 2D to 3D video conversion, depth maps, image segmentation, semi-automatic, graph cuts, random walks, object tracking, motion estimation, computer vision

Copyright © 2013 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org

R. Phan and D. Androutsos are with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, Ontario, M5B 2K3, Canada. (e-mail: rphan@ee.ryerson.ca, dimitri@ee.ryerson.ca)

Editors Information Classification Scheme (EDICS): 2-PRES
I. INTRODUCTION

While stereoscopic imaging has been around for many decades, it has recently seen a rise in interest due to the availability of consumer 3D displays and 3D films. This surge is seen in the recent proliferation of big budget 3D movies, or with some portions of the film being offered in 3D. 3D films show novel views of the scene for the left and right eyes to the viewer. Humans view the world using two views, which are processed by the visual cortex in our brain, and we thus perceive depth. This enhanced experience by exploiting depth perception is the main attraction for viewing 3D films.

While much of the initial focus in stereoscopic 3D was on creating display hardware, attention has now shifted to efficient methods of creating and editing stereoscopic 3D content. 3D films are subject to a number of constraints arising from creating the distinct views. For example, the size of the viewing screen can have an impact on the overall perceived depth. If any depth errors surface, they may not be apparent on a mobile screen, but may be so on a cinema screen. There have been remapping techniques to ensure a consistent perception of depth, such as the method by Lang et al. [1]. They propose four simple operators for remapping, but they do not consider relative depth changes among neighboring features, and also content preservation, such as planes, and left/right frame coherence. This is successfully addressed in Yan et al. [2], and was shown to outperform [1]. For 2D to 3D conversion, the current accepted method is a mostly manual labor intensive process known as rotoscopy. An animator extracts objects from a frame, and manually manipulates them to create the left and right views. While producing convincing results, it is a difficult and time consuming, inevitably being quite expensive, due to the large requirement of manual operators. Despite these problems, this is a very important part of the stereoscopic post-processing process, and should not be dismissed. 2D to 3D conversion of single-view footage can become useful in cases where filming directly in 3D is too costly, or difficult. Research into conversion techniques is on-going, and most currently focus on generating a depth map, which is a monochromatic image of the same dimensions as the image or frame to be converted. Each intensity value represents the distance from the camera - the greater/darker the intensity, the closer/farther the point in the image or frame is to the camera. Novel viewpoints can be generated from this depth map using Depth Image Based Rendering (DIBR) techniques [3]. An example of what a depth map can look like is shown in Fig. 1 below.
A. Related Work

Recent research in 2D to stereoscopic 3D conversion include the following, each concentrating on a particular feature or approach to solve the problem.

1) Motion: The idea is that for objects closer to the camera, they should move faster, whereas for objects that are far, they should be slower. Motion estimation can be used to determine correspondence between two consecutive frames, and to determine what the appropriate shifts of pixels are from the one frame to the next. Wang et al. \cite{4} perform a frame difference to eliminate static backgrounds, and use block-based motion estimation to infer depth, with some post-processing done using morphological operators. In Liu et al. \cite{5}, each shot is classified as either being static, or one comprised of motion. Static scenes use geometric transformations while for a motion scenes, motion vectors are employed with post-processing to smooth and enhance edges. Finally, with Chen et al. \cite{6}, they fuse two depth maps together: motion and color. Motion employs the magnitude, while color employs the Cr channel from the YCbCr color space.

2) Using Scene Features: Any method involving the direct use of features, such as shape and edges, are of interest here. In Kuo et al. \cite{7}, the objects within the scene are classified into one of four types: sky, ground, man-made and natural objects. An initial depth is assigned to each object using vanishing points, a simplified camera projection model, and object classification algorithms. In Zhang et al. \cite{8}, the depth maps are estimated in a multi-cue fusion manner by leveraging motion and photometric cues in video frames with a depth prior of spatial and temporal smoothness. Finally, Schnyder et al. \cite{9} create an automatic system for use in field-based sports by exploiting context-specific priors, such as the ground plane, player size and known background.

3) Other Conversion Methods: An example of a technique not using any of the aforementioned methods is by Konrad et al. \cite{10}. Among millions of stereopairs available on the Internet, there likely exist many stereopairs whose 3D content matches the image of interest. They find a number of online
stereopairs whose left image is a close photometric match to the query and extract depth information from these stereopairs. Finally, for Park et al. [11], the goal is to identify the content within the video that would have a high quality of depth perception to a human observer, without recovering the actual 3D depth. They use a trained classifier to detect pairs of video frames, separated by time offsets, that are suitable for constructing stereo images.

B. Our Method of Focus: Semi-Automatic

At the present time, most conversion methods focus on extracting depth automatic automatically. However, even with minimal user intervention, these can become extremely difficult to control conversion results - any errors cannot be easily corrected. No provision is in place to correct objects appearing at the wrong depths, and may require extensive pre-/post-processing. Therefore, there are advantages to pursuing a user-guided, semi-automatic approach: For a single image or frame, the user simply marks objects and regions to what they believe is close or far from the camera, denoted by lighter and darker intensities respectively. One question often arising is whether or not the depths that the user mark are indeed the right ones for perception in the scene. However, the depth labeling need not be accurate; it only has to be perceptually consistent, and quite often, the user labeling will meet this criteria [12]. The end result is that the depths for the rest of the pixels are estimated using this information. For the case of video, the user marks certain keyframes, each in the same fashion as the single image, and the rest of the depths over the video are estimated. By transforming the process into a semi-automatic one, this allows the user to correct depth errors that surface, should they arise. This ultimately allows for faster and more accurate 2D to 3D conversion, and provides a more cost-effective solution.

There are some methods well known in this realm. In particular, the work by Guttmann et al. [12] solves for the depths in a video sequence by marking strokes only on the first and last frames. The first step is to detect anchor points over all frames. A rough estimate of the depth for the first and last frame is found, and Support Vector Machine (SVM) classifiers are trained on both frames separately. Each SVM is trained on classifying a unique depth in the rough estimate, using the Scale-Invariant Feature Transform (SIFT), combined with the grayscale intensity at the particular point as the input space. To detect anchor points, SIFT points are detected throughout the entire video, and are put through the SVM classifiers. For each SIFT point, a One-Vs-All scheme is employed, choosing the highest similarity out of all the classifiers. If this surpasses a high-confidence threshold, it is an anchor point, and the depth assigned to it is the one belonging to the SVM classifier tuned for the particular depth. To solve for the rest of the depths, an energy function is minimized by least squares. The function is a combination of
spatial and temporal terms, color information, the anchor points and user strokes. The minimization is performed by transforming the problem into a sparse linear system of equations, and directly solved. A similar method was created by Wang et al. [13], where holes / occlusions are handled quite well; they are simply snapped together with nearby non-occluding areas. However, the issue with [12] is that it is quite computationally intensive. Not only does this require SIFT points and SVM classifiers to be computed, but it requires solving a large set of linear equations, requiring a large amount of memory.

Upon closer inspection, we can consider semi-automatic 2D to 3D conversion as a multi-label image segmentation problem. Each depth can be considered as a unique label, and to create a depth map, each pixel is classified to belong to one of these depths. Inspired by Guttmann et al., we propose a stereoscopic conversion system to create depth maps for 2D to 3D conversion. The core of our system combines the merits of two semi-automatic segmentation algorithms to produce high quality depth maps: Random Walks [14] and Graph Cuts [15]. In the segmentation viewpoint, automatic methods are not an option, as we wish to leverage the user in creating accurate results, and would also be contrary to our viewpoint on automatic 2D to 3D conversion. There are many semi-automatic algorithms in practice. Examples are Intelligent Scissors [16], or IT-SNAPS [17], where the user clicks on pixels, known as control points, that surround an object of interest to extract. In between the control points, the smallest cost path is determined that can delineate the foreground object from the background. However, this can only be performed on objects that have a clear separation from the background. In addition, this method could not be used in specifying depths on background areas, as the core of the algorithm relies on separation between the background and foreground by strong edges. Another example is the Simple Interactive Object Extraction (SIOX) framework [18], which operates in the same way as Graph Cuts and Random Walks with the use of foreground and background strokes, but unfortunately only works on single images. We ultimately chose Random Walks and Graph Cuts, not only because marking regions of similar depth is more intuitive, but these can ultimately be extended to video sequences.

For the case of single images, producing a depth map is akin to semi-automatic image segmentation. However, video can become complicated, as ambiguities in designing the system surface, such as what keyframes to mark. It is natural to assume that marking several keyframes over video is time consuming. In this aspect, we allow the user the choice of which keyframes to mark, as well as providing two methods for automatically propagating the strokes throughout the sequence. These methods will be discussed in Section II-C, each having their merits and drawbacks, but are chosen depending on the scene at hand. In this paper, we first discuss the general framework for the conversion of images. After, we consider video, investigating methods for minimizing user effort in label propagation. We also show sample depth
II. METHODOLOGY

Our framework relies on the user providing an initial estimate of the depth, where the user marks objects and regions as closer or farther from the camera. We allow the user to mark with monochromatic intensities, as well as a varying color palette from dark to light, serving in the same function as the intensities. A flow chart illustrating our framework is seen in Fig. 2.

A. Graph Representation

To ensure the highest level of flexibility, we treat both images and video the same - as $N$-connected graphs. A graph in this viewpoint can be considered as a lattice of nodes, where each pixel is one node. The pixels, or nodes, have edges connecting them together. The edges are assigned numerical values, representing how similar the connected pixels are from each other. What measures the similarity can be a variety of different entities, and will be discussed later. An $N$-connected graph denotes $N$ edges leaving each node to connect to other nodes. Fig. 3 is an example of a $4 \times 4$ image labeled with three depths with different colors.

Each node is a pixel that connects to its vertical and horizontal neighbors, and thus it is a 4-connected graph, which is what we use in our framework for the sake of memory constraints. In the case of video,
it is a three-dimensional graph, where each pixel in one frame not only connects with nodes within the same frame (spatially), but also temporally. In our framework, for the sake of memory constraints, this is a 6-connected graph: a frame is connected in a 4-way fashion, in addition to the node in the same spatial location in the previous frame, as well the next frame.

B. Conversion Framework for Images

This graph representation has been successfully used in image segmentation, most notably with Graph Cuts [15][19], and Random Walks [14]. We use a combination of Graph Cuts and Random Walks to find an optimal labeling of the depths for the source images or videos, given an initial set of depth labels / strokes. Graph Cuts is used to determine an initial depth map, known as a depth prior [20]. This serves as an additional channel of information into the Random Walks algorithm, and is weighted accordingly to control the contribution this information has to the final output. The practical reasons of combining these two methods will become evident later, but we first discuss how we generate the depth maps for each framework separately.
1) Random Walks for Images: Random Walks \cite{14} is an optimization scheme that finds the likelihood of a random walker, starting at some label, visiting all of the unlabelled nodes in the graph. The walker is biased by the edge weights so that it is more likely to visit similar nodes than dissimilar ones. For the purpose of image segmentation, the goal is to classify every pixel in an image to belong to one of \( K \) possible labels. Random Walks determines the probability of each pixel belonging to one of these labels, and the label that the pixel gets classified as is the one with the highest probability. This is performed by solving a linear system of equations, relying on the Laplacian Matrix. If \( v_i \) represents the \( i \)th pixel in the image, that is of size \( R \times C \), then the Laplacian matrix, \( \mathcal{L} \), of size \( RC \times RC \) is defined in Eq. 1 as the following:

\[
\mathcal{L} = \forall (i,j) \in \mathcal{P} \text{s.t. } \begin{cases} 
\deg(v_i) & \text{if } i = j \\
-1 & \text{if } i \neq j \text{ and } v_i \text{ is connected to } v_j \\
0 & \text{otherwise}
\end{cases}.
\]  

(1)

(\( i,j \)) are pixels in the image \( \mathcal{P} \), and \( \deg(v_i) \) is the degree of the pixel \( i \), which is the sum of all of the edges leaving \( i \). Each row of \( \mathcal{L} \) is an indication of how each pixel \( i \) is connected to each other. Additionally, let us define the vector \( \vec{x} \), of size \( RC \times 1 \), where the \( i \)th row is the probability that the pixel \( i \) will be assigned a particular label, \( k \). By letting \( Q(v_i) \) represent the user-defined label of the \( i \)th pixel, for a particular user-defined label \( k \), vector \( \vec{x} \) is defined as:

\[
\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_{RC} \end{bmatrix}^T, \text{ s.t. } x_i = 1, \text{ if } Q(v_i) = k, \text{ and } x_i = 0 \text{ if } Q(v_i) \neq k, \forall i = 1,2,\ldots,RC.
\]  

(2)

For those pixels not user-defined (i.e. those that have no \( Q(v_i) \)), those are the unknown probabilities that need to be solved. If we re-arrange the vector in Eq. 2 into two sets, such that those pixels marked by the user appear first, \( x_M \), followed by those unknown after, \( x_U \), we obtain a decomposition of \( x = [\vec{x}_M \vec{x}_U]^T \).

By keeping track of how we rearranged the vector \( x \), by performing the same rearranging with the rows of \( \mathcal{L} \), we thus obtain the following decomposition for \( \mathcal{L} \):

\[
\mathcal{L} = \begin{bmatrix} \mathcal{L}_M & B^T \\
B & \mathcal{L}_U \end{bmatrix}
\]  

(3)

Finally, to solve for the unknown probabilities for the label \( k \), we solve the matrix equation of \( L_U \vec{x}_U = -B^T \vec{x}_M \) \cite{14}. We construct \( \vec{x} \) for each label \( k \), and solve for the unknown probabilities of \( k \). Whatever
label gives the maximum probability for a pixel in \( x_U \), that is the label it is assigned. For use in generating depth maps, we modify this methodology in the following way. The probabilities within Random Walks spans between the real set of \([0, 1]\). Therefore, we allow the user-defined depths, and ultimately the solved depths for the rest of the image or frame to be from this set, allowing for “manually” setting the probabilities of the marked pixels within the vector \( \vec{x} \). The goal is to solve for only one label, the \textit{depth}. As such, the user chooses values from the set of \([0, 1]\) to brush over the image or frame, where 0/1 represents a dark/light colour or intensity. After, the rest of the depths are solved by Eq. 3. The resulting probabilities can directly be used as the depths for generating stereoscopic 3D content. As a means of increasing accuracy, and to combat noise, we employ Scale-Space Random Walks (SSRW) [21], which samples the image by a multi-resolution pyramid. Random Walks is applied to each scale within the pyramid, upsampled, and are merged using the geometric mean. For the edge weights, the dissimilarity function is one that is frequently used in pattern classification: the Sigmoidal function. Between two pixels, \( v_i \) and \( v_j \), it is defined in Eq. 4 as:

\[
N(v_i, v_j) = \gamma \left( \frac{2}{1 + \exp \left( \beta D(\vec{c}_i, \vec{c}_j)^\alpha \right)} \right).
\] (4)

\( D(\vec{c}_i, \vec{c}_j) \) is the Euclidean distance of the CIE L*a*b* components between \( v_i \) and \( v_j \), and \( \alpha, \beta, \gamma \) are parameters controlling how dissimilar two colors are. These were experimentally set to \( \alpha = \gamma = 1 \), and \( \beta = 2 \).

Unfortunately, Random Walks comes with certain issues. A consequence with specifying depths in a continuous range is that it is possible that depths will be produced that were not originally specified. This is certainly the desired effect, allowing for internal depth variation for objects, and would eliminate objects being perceived as “cardboard cutouts”. In terms of weak contrast, this results in a “bleeding” effect, where regions at a closer depth can bleed into regions of farther depth. We show an example of this in Fig. 4, which is a snapshot of an area on the Ryerson University campus. Fig. 4(a) shows the original image, while Fig. 4(b) shows the user-defined scribbles, and Fig. 4(c) is the depth map produced by SSRW. The user only needs to mark a subset of the image, and a reasonable depth map is produced. Though there is good depth variation on the objects and background, there is bleeding around some of the object boundaries. This is unlike Graph Cuts, which provides a hard segmentation. The depth map generated only consists of the labels provided by the user.

2) \textit{Graph Cuts for Images}: Graph Cuts is based on solving the Maximum-A-Posteriori Markov Random Field (MAP-MRF) labeling problem with user-defined or hard constraints [19]. The solution to
the MAP-MRF labeling problem is to find the most likely labeling for all pixels from the provided hard constraints. It has been shown in [19] that the solution can be found by minimizing the following energy function:

\[ E(\mathcal{P}) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q). \] (5)

Here, \( \mathcal{P} \) is the set of pixels making up an image, and \( p \) is a pixel from the image. \( E(\mathcal{P}) \) is the “energy” of the image. \( D_p(f_p) \) represents the data cost, or the cost incurred when a pixel \( p \) is assigned to the label \( f_p \). \( V_{p,q}(f_p, f_q) \) is the smoothness cost, or the cost incurred when two different labels are assigned to two different pixels within a spatial neighborhood \( \mathcal{N} \), which is a 4-connected graph in our framework. Referencing graph theory, it has been shown that the solution that minimizes \( E(\mathcal{P}) \) is the solution that finds the max-flow/min-cut of a graph [19]. Efficient algorithms and software have been created to perform this minimization, and is available by referring to [19].

As depth map generation can be considered as a multi-label classification problem, but Graph Cuts only focuses on the binary classification problem. As such, for each pixel, there are foreground and background costs associated with them. Therefore, each unique user-defined depth value is assigned an integer label from the set \( B \in [1, N_D] \), where \( N_D \) represents the total number of unique depths present in the user-defined labeling. A binary segmentation is performed separately for each label \( b \in B \). The user-defined labels having the label of \( b \) are assigned as foreground, while the other user-defined labels serve as background. The rest of the pixels are those we wish to label. Graph Cuts is run for a total of \( N_D \) times, once for each label, and the maximum flow values for the graph are recorded for each label in \( B \). If a pixel was only assigned to one label, then that is the label it is assigned. However, if a pixel was assigned multiple labels, we assign the label with the highest maximum flow, corresponding to the least amount of energy required to classify the pixel. In some cases, even this will not always result in every pixel being classified, but region filling methods can be used to correct this.

In our Graph Cuts framework, we use the Sigmoidal function (Eq. 4) to reflect the smoothness costs. For the data costs, we use a modified Potts model, rather than the negative log histogram probability as done in [15]. The reason why was due to the nature of the color space to represent images. The negative log histogram probability is computed in the RGB color space, which is inherently discrete. However, we wish to employ the CIE \( L^*a^*b^* \) color space, as it naturally captures the human perception of color better than RGB. In addition, CIE \( L^*a^*b^* \) has floating point values for each of their components, and so it is not intuitive to determine how many bins are required when creating a histogram. Table I shows
how the data costs are computed when the user-defined label $b \in B$ is the one chosen as foreground. In
the table, the chosen foreground label during a particular Graph Cuts minimization is denoted as $m$, and $K$ is defined as the following [15].

$$K = 1 + \max_{p \in P} \sum_{q \in \{v_p, v_q\}} N(v_p, v_q).$$  \hspace{1cm} (6)

<table>
<thead>
<tr>
<th>Type of Link</th>
<th>$m = b, b \in B$</th>
<th>$m \neq b, b \in B$</th>
<th>$m \neq b, b \notin B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Link</td>
<td>$K$</td>
<td>0</td>
<td>$K$</td>
</tr>
<tr>
<td>Sink Link</td>
<td>0</td>
<td>$K$</td>
<td>$K$</td>
</tr>
</tbody>
</table>

TABLE I: Data costs used in the Graph Cuts segmentation for a label $b$, when the chosen foreground label is $m$.

Fig. 4(d) illustrates a depth map obtained using only Graph Cuts, using the same labels in Fig. 4(a) to be consistent. As with Random Walks, only certain portions of the image are labeled, and a reasonably consistent depth map was still generated successfully. It should be noted that there are some portions of the image that failed to be assigned any labels, such as the area around the right of the garbage can. However, this can certainly be rectified by using region-filling methods. Though the result generated a depth map respecting object boundaries very well, the internal depths of the objects, as well as some regions in the background do not have any internal depth variation. If this were to be visualized stereoscopically, the objects would appear as “cardboard cutouts”. Noting that Random Walks creates smooth depth gradients, and combining this with the hard segmentation of Graph Cuts, Random Walks should allow for depth variation to make objects look more realistic, while Graph Cuts eliminates the bleeding effect and respect hard boundaries.

3) Merging the two together: To merge the two depth maps together, a depth prior is created, serving as an initial depth estimate, and provides a rough sketch of the depth. This information is fed directly into the Random Walks algorithm. The depth prior is essentially the Graph Cuts depth map, and should help in maintaining the strong boundaries in the Random Walks depth map. Before merging, we must modify the depth prior. The depth map of Random Walks is in the continuous range of $[0, 1]$, while the depth map for Graph Cuts is in the integer range of $[1, N_D]$. Both depth maps correspond to each other, but one needs to be transformed into the other to both be compatible. As such, when the Graph Cuts depth map is generated, it is processed through a lookup table $T[\downarrow]$, where $\downarrow$ is an integer label in the Graph...
Cuts depth map. $T[\mathbb{I}]$ transforms the integer labeling into a compatible labeling for Random Walks. This is done by first creating a labeling within $[0,1]$ for Random Walks, and to sort this list into a unique one in ascending order. Each depth in $T[\mathbb{I}]$ is assigned an integer label from $[1,2,\ldots,N_D]$, keeping track of which continuous label corresponds to which integer label. After Graph Cuts has completed, this correspondence is used to map back to the continuous range.

To finally merge the two depth maps together, the depth prior is fed into the Random Walks algorithm by modifying the edge weights directly. The depth prior is incorporated by performing a Random Walks depth map generation, but we modify the edge weight equation (Eq. 4), where the distance function is appended to include information from the depth prior. If we let $d_i$ represent the depth from the depth prior at pixel $i$, after being processed through the lookup table, the edge weight equation is modified by modifying the Euclidean distance, which is defined as:

$$D(\vec{c}_i, \vec{c}_j, d_i, d_j|\alpha) = \sqrt{[D(\vec{c}_i, \vec{c}_j)]^2 + \alpha(d_i - d_j)^2}.$$  \hspace{1cm} (7)

$\vec{c}_i, \vec{c}_j$ and $D(\vec{c}_i, \vec{c}_j)$ are defined in section II-B1. $\alpha$ is a positive real constant, which we set to 0.5, serving as a scaling factor determining how much contribution the depth prior has to the output. $d_p$ denotes the depth at pixel $p$ from the depth prior. However, the dynamic range of the depth values from the depth prior is within $[0,1]$, where as the components of the CIE L*a*b* color space, as seen in Eq. 7, are significantly larger. As such, these terms will overpower the terms in the depth prior. With this, the entire image / frame is converted into the CIE L*a*b* color space first. Each CIE L*a*b* channel is normalized individually, so that all components for each channel is within $[0,1]$, and are the ones used in Eq. 7.

Fig. 4(e) shows an example of the merged results, where we set $\alpha = 0.5$. When compared to Figs. 4(c) and 4(d), Fig. 4(e) contains the most desirable aspects between the two. The depth prior has consistent and noticeable borders for the objects in the scene, while the Random Walks depth map alone contains subtle texture and gradients to those objects. The trees and shrubbery in the original image are now much more differentiated from the background and neighboring objects than before. In addition, the areas that were left unclassified in the depth prior, or the “holes”, have been filled in after the final depth map was produced. The reason is because the edge weights from Random Walks ultimately considered the depth prior as only one component of information out of the overall feature vector used, and can still rely on the color image data to produce the final depth map.
C. Conversion Framework for Video

Video is essentially a sequence of images at a specified frame rate. When generating depth maps, our framework operates in the same fashion, except that there are two fundamental areas that need to be modified: (a) Instead of labeling one frame, several keyframes must be labeled. (b) The graph representation needs to be modified from a 4-connected graph to a 6-connected graph. This ensures that the frames connect together in a temporal fashion, and the depth map in one frame is consistent with the next frame. Before explaining the mechanics of how video conversion is performed, we are assuming that there are no abrupt changes in the video being processed, or consisting of a single camera shot. If a video has any camera changes, then it must be split up into smaller portions, with each not having a change of shot. This can be done either manually, or using an automated shot detection system.

1) Keyframes to label: One question that a user may ask is how many frames are required for labeling in order for the depth map to be the most accurate? One option is to allow the user to manually choose the frames they wish to label, and this is an option we allow in our system. However, for the video segmentation realm, at least the first and last frames need to be labeled, in order for the segmentation to be consistent across the sequence. This is the approach taken in Guttmann et al.’s work. However, while it is possible to label only certain keyframes, this will result in a number of different depth artifacts. Fig. 5 illustrates a sample shot, with their labels overlaid. This comes from the Sintel sequence\(^2\), and this shot is 38 frames. Fig. 5 is comprised of the 1\(^{st}\), 19\(^{th}\) and 38\(^{th}\) frames, with their labels superimposed. Fig. 6 illustrates some depth maps within the sequence using our framework, showing what happens when only frames 1, 19 and 38 are labelled. Finally, Fig. 7 illustrates what happens when all frames are labeled in the same style as Fig. 6, again using our framework to video. In both Figs. 6 and 7, the first row illustrates frames 1 and 13, while the second row illustrates the frame 25 and 38. For the sake of brevity, we only show the labels for frames 1, 19 and 38. The color scheme for the labels is the same as in Fig. 4.

As can be seen, for the frames that had no labels, the rapidly moving parts quickly “fade” in depth or appear to move away from the camera. This is clearly not the case, since all objects in the scene move horizontally, and not farther away. If all of the frames are labelled appropriately, then the depth remains consistent for all frames. While labeling every frame produces the best results, it is not the easiest task to perform. For a modest sequence, such as the 38 frame Sintel sequence, labeling each frame manually is quite a tedious task, and would take a considerable amount of time. However, if
the labeling can be performed in a more automated fashion, this would simplify the task of creating a more detailed labeling, and thus increase the accuracy of the depth maps. The idea is to only label the first frame with the user-defined depth labels. After, these labels are tracked throughout the entire video sequence. In addition, as the video sequence progresses in time, it will contain areas where the depth will need to change from what the user defined those areas to be. As such, there has to be a way of dynamically adjusting those depths when those situations arise. As mentioned earlier in Section I, we reference two methods for tracking these labels, and depending on the scene at hand, we perform this automatic labeling is through using one of two computer vision based algorithms: (a) An algorithm for tracking objects in unconstrained video, since unconstrained is the type that our framework will most likely encounter, and (b) An optical flow based algorithm that is employed, should the scene exhibit only horizontal motion and can avoid some of the computational burden exhibited in (a). What is attractive about (b) is that it determines optical flow without the use of global optimization. This becomes very...
attractive for implementation in dedicated parallel architectures, such as GPUs or FPGAs.

2) Label Tracking - Object Tracking Approach: As user-defined depth strokes will inevitably vary in shape and length, it would be computationally intensive to track all points within a stroke. As such, the proposed system considers a single stroke with a given user-defined depth as an ordered set of $N$ points. As such, in our framework, each stroke is thinned using binary morphological thinning, to reduce the stroke to a minimally connected one, and is uniformly subsampled. If the user draws overlapping strokes at the same depth, a clustering is performed to combine these to a single stroke. Once all points for all strokes are collected, they are individually tracked, and their depths are automatically adjusted when necessary. Cubic spline interpolation is applied after, to connect these control points to approximate the final strokes with high fidelity, constrained to the locations of the tracked points. To reconstruct the width of the stroke after thinning, the smallest distance between center of mass of the thinned stroke and the perimeter of the original stroke is found. This overall process inevitably creates “user”-defined strokes across all frames, reducing the amount of interaction involved.

To robustly track these points, we use a modification of one created by Kalal et al. [22], known as the Tracking-Learning-Detection framework (TLD). The user only draws a bounding box around the object of interest in the first frame, and its trajectory is determined over the entire video sequence. Essentially, bounding boxes of multiple scales are examined in each frame to determine if the object is present. However, the tracker is optimally designed to track objects. When the depth strokes coincide to background, these are handled differently, and will be discussed later. If more than one frame is marked,
the tracker is restarted, using the strokes in each user-marked frame as the “first” frame.

The initial location of the object is tracked using a multi-scale pyramidal KLT [23], and the location is further refined using a two-stage detector. This is comprised of a randomized fern detector (RFD) [24] and an intensity-based nearest neighbor classifier (NNC), resizing the pixels within a bounding box to a $15 \times 15$ patch, with normalized cross-correlation as the similarity measure. The KLT, as well as the RFD and NNC, operate on monochromatic frames, and are converted accordingly if the input is in color. However, it has been noted that color images are naturally acquired in the RGB color space, with significant correlation between each color component. Information from other channels could have contributed to the acquisition of the target channel, and simply converting to grayscale ignores this correlation [25]. However, the KLT, as well as the RFD can achieve good results ignoring color information. In retrospect, the NNC can most definitely be improved on, as this can use color information, and can be transformed into a more perceptually uniform color space so that quantitative differences in colors reflect what is perceived by human beings.

We thus modify the NNC framework to incorporate the similarity between two color patches using Phan et al.’s framework [25], with color edge co-occurrence histograms (CECH). These are three-dimensional histograms, indexing the frequency of pairs of colors lying on valid edges over different spatial distances; the first two dimensions are colors, and the third is the spatial distance. Some quantization of colors needs to be performed to allow for feasible memory storage, and Phan devised a method that captured human color perception and recall well, and is illumination invariant. Ultimately, the NNC was adapted to use the similarity values from these histograms to increase overall accuracy of detecting color distributions within the bounding boxes, and are based on the correlation between pairs of histograms. Further details on how these descriptors were computed, and how they are adapted into the TLD framework can be found in [26].

In order to dynamically adjust the depth of objects, the ability of TLD to detect objects over multiple scales is used. Let $B = \{b_0, b_1, \ldots, b_i, \ldots, b_{M-1}\}$ represent the scales of the initial bounding box of size $R \times C$ created in the first frame, such that for a scale $b_i$, the bounding box has dimensions $b_i R \times b_i C$. When TLD tracking is performed, as the object moves closer/farther to the camera, the size of the bounding box required for the object will increase/decrease, as its perceived size will increase/decrease. This kind of motion will most likely be non-linear in nature, and so a mapping function is created, describing the relationship between the scale of the detected bounding box, with the labelled depth to be assigned for that point; a simple parabolaic perturbation is opted for use. If $D(z)$ represents the mapping function for a scale $z \in B$, a parabolic relationship can be determined, where $D(z) = az^2 + bz + c$. With TLD, two
details are already known. The smallest scale \( b_0 \) is the farthest depth, \( d_0 \), and the largest scale \( b_{M-1} \) is the closest depth, \( d_{\text{max}} \), and are 0 and 255 respectively in our framework. Finally, the initial bounding box has a scale of 1.0, with the depth assigned from the user, \( d_u \), and the coefficients are solved by the inverse of the following system:

\[
\begin{bmatrix}
(b_0)^2 & b_0 & 1 \\
1 & 1 & 1 \\
(b_{M-1})^2 & b_{M-1} & 1
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c
\end{bmatrix}
=
\begin{bmatrix}
d_0 \\
d_u \\
d_{\text{max}}
\end{bmatrix}
\]  

(8)

At each frame, the bounding box scale automatically adjusts the depth at this point along a stroke. This relationship between the scales and depths functions quite well; if the current bounding box is of the original scale, object will have only exhibited horizontal motion. As the object moves closer and farther from the camera, the depth is automatically adjusted within this bounding box. This will function well when different parts of the object appear at different depths (i.e. when the object is perceived at an angle). The TLD framework, and ultimately ours, is designed to track objects. As the goal of the proposed work is to determine dense depth maps, there will inevitably be strokes placed on the background, and not on objects. As the TLD tracker is also comprised of the KLT, if the stroke lies on a uniform background, the online learning, RFD and NNC should be disabled, and the KLT should only function. To determine when to only enable KLT, we use the saliency detection algorithm by Goferman et al. [27] which effectively determines salient regions, and captures nearby pixels to faithfully represent the scene. For each bounding box centered at a stroke point, the mean saliency value is computed. If it surpasses a normalized threshold, this is considered salient, and the full TLD tracker is used. Otherwise, only KLT is used for tracking. In this work, a threshold of 0.7 is used.

3) Label Tracking - Optical Flow: Though the modified TLD tracker is robust, it can become computationally intensive. There is an additional computational step using a saliency detector, and additionally, a set of randomized fern detectors is required to be trained: one per stroke point. Noting this drawback, we offer an alternative method that the user may choose. This is through the use of optical flow, and we opt to use a paradigm that avoids using global optimization to encourage implementation in parallel architectures, and modify this so that the framework is temporally consistent across all frames in the video. Optical flow seeks to determine where pixels have moved from one frame to the next frame. As such, to perform label propagation, one simply needs to move pixels that coincide with the user-defined strokes by the displacements determined by the optical flow vectors, and this is exactly what is done in our framework. Most traditional optical flow algorithms calculate the flow between two consecutive
frames, and there may be information over neighboring frames of video that may improve accuracy, thus leading to our desire for temporal consistency. It should be noted that this approach is suitable only for motion and objects that are moving horizontally. Currently, we have not developed a way of dynamically adjusting the depths of the objects using optical flow like in Section II-C2, but is currently an ongoing area of research being pursued.

The framework that we use is based on the one by Tao et al. [28], known as SimpleFlow, which is a non-iterative, multi-scale, sub-linear algorithm. However, SimpleFlow only considers local evidence, and scenes with fast motion will inevitably fail. As such, we augment SimpleFlow by introducing edge-aware filtering methods that directly operate on the optical flow vectors, and consider information over neighboring frames to further increase the accuracy. In most edge-aware filtering methods, the filtered outputs are generated by considering the content of a guidance image, which serves as additional information to the filtering mechanism that the output must satisfy. We can thus improve accuracy by using the frames in the video sequence as guidance images. The flows for each frame can be filtered independently, but in order for the vectors to be temporally consistent, we also filter temporally. As the optical flow vectors are only computed between two consecutive frames independently, it is inevitable that motion will be non-smooth. As such, edge-aware filtering on a temporal basis, with frames as guidance images will inevitably allow smoothness in the flow vectors across the video, and become temporally consistent.

The edge-aware filtering method we use is from Gastal and Oliveira [29]. In [29], distance-preserving transforms are the main tool for edge-aware filtering. The idea is that if an image is transformed into another domain, such that similarity values in the original domain are the same in the transformed domain - one that is a lower dimensionality, filters designed in the transformed domain will be edge preserving, and is known as the domain transform. However, the formulation is only for 1D signals. To filter 2D signals (i.e. images), [29] show that no 2D domain transform exists, so approximations must be made, leading to iterations. Each row is independently filtered, then each column is filtered immediately after. Using the image frames as guidance images should mitigate any errors in the flow maps due to occlusions, as their edge information will “guide” the filtered vectors. However, the current iteration mechanism does not address temporal consistency. To address this, one complete iteration consists of a 2D filtering operation, with the addition of a temporal filtering operation. However, one must be mindful when filtering temporally. The optical flow vectors are used to determine where points in one frame map to the next consecutive frame. To determine these, one simply needs to take the every spatial co-ordinate in the one frame, and use the vectors to determine where these pixel locations are best located in the
By doing this, we can determine a binary map for each frame, $B$, where a location is set to 1 if an optical flow vector maps a pixel to this location in the next frame, and 0 otherwise. This binary map determines those locations where the optical flow is reliable, and ensuring that the flow vectors are temporally consistent. In order to filter temporally, we apply the same technique performed on 1D signals in [29] on a temporal basis. Specifically, in a video sequence of $K$ frames, there will be an optical flow vector at a particular location $(x, y)$ at each frame. Each location $(x, y)$ will thus have a set of $K$ optical flow vectors - one per frame. For a video sequence with a frame size of $M \times N$, we will thus have $MN$ sets of $K$ optical flow vectors, and each set is a temporal 1D signal. These signals are filtered using the method in [29]. For a particular frame in the sequence, if a location of $B$ in this frame is 1, then the domain transform is computed normally. However, should the location be 0, the domain transform is computed by using information from the previous frames, as this information is more reliable before this particular frame. To further increase accuracy, we introduce an occlusion term, so that those locations that are likely to be occluded do not contribute substantially to the output. This occlusion term is calculated by creating a confidence map that uses the forward flow and backward flow vectors, in conjunction to a sigmoidal function to ultimately decide how confident the optical flow vectors are throughout the scene. Further details regarding the overall process of achieving temporally consistent optical flow can be found in [30].

III. Experimental Results

In this section, we will demonstrate depth map results using our image and video conversion framework. For images, these are shown only for demonstration purposes, as the user has the option of converting solely images. We will show some depth map results using a variety of different media and sources, as well as their corresponding anaglyph images so that the reader has a sense of our framework works for images. However, the main focus of our work is converting video sequences. For the video sequences, we show depth maps for subsets of the frames in each sequence, as well as their anaglyph images. We also compare to Guttmann et al.’s work and show their depth maps and anaglyph images on the same videos, using the same user-defined strokes to be consistent in comparison.

A. Results: Images

Figs. 8, 9 and 10 shows a variety of examples from all areas of interest. For each figure, the first column shows the original images, with their user-defined depth labels overlaid, the second column shows their depth maps, and the last column shows the anaglyph images, illustrating the stereoscopic versions of the
Fig. 8: The first set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images.

The anaglyph images are created so that the left view is given a red hue, while the right view is given a cyan hue. To create the stereoscopic image, the left view serves as the original image, while the right view was created using simple Depth Image Based Rendering (DIBR). To perform DIBR, the depths are used as disparities, and serve as translating each pixel in the original image over to the left by a certain amount. Here, we introduce a multiplicative factor, $s$, for each value in the depth map, and introduce a bias, $b$, to adjust where the convergence occurs in the image. The purpose of $s$ is to adjust the disparities for the particular viewing display of interest. The actual depths in the depth maps are normalized in the range of $[0, 1]$, and we render the right view using the aforementioned process. Given a value in the original depth map, $D_O$, the equation used to determine the shift, $F_S$, of each pixel from the left view, to its target in the right, is $F_S = sD_O + b$. 
Fig. 9: The second set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images.

In all of our rendered images, as well as the videos, we chose $s = 30$, and $b = -15$ so that the disparities can clearly be seen, given the limited size of figures for this paper. In addition, we request that the reader zoom into the figures for the fullest stereoscopic effect. To demonstrate the full effect of labelling with different intensities or different colors, the figures vary in their style of labelling, and use either: (a) A grayscale range, varying from black to white (b) Using only the red channel, varying from dark to light red, and (c) A jet color scheme, where black and red are the dark colors, while yellow and white are the light colors.

Fig. 8a shows a frame of the Avatar movie, while Figs. 8b and 8c illustrate the depth map and the stereostopic version in anaglyph format. The scene requires very few strokes in order to achieve good depth perception, as evidenced by the depth map. The edges around the prominent objects, such as the
Fig. 10: The third set of conversion examples using various images from different sources of media. Left Column - Original images with depth labels overlaid. Middle Column - Depth maps for each image. Right Column - Resulting anaglyph images

hand, the levitating platform, and the rock are very smooth, which will minimize the effect of “cardboard cutouts” upon perception on stereoscopic displays. Also, along the ground plane, there is a smooth transition near the front of the camera, all the way to the back as expected. The same can be said with Figs.8d-8f and Figs. 8g-8i. These are images taken from downtown Boston, and only a few strokes were required to achieve a good quality depth map. Like the Avatar image, there are smooth gradients from the front of the camera to the back, and the edges are soft so that perception on stereoscopic displays will be comfortable. The images of Fig. 8 have also been chosen on purpose, as there are prominent objects in the foreground, while there are other elements that belong to the background. By assigning these objects to have depths that are closer to the camera, while the background has depths assigned to those that are farther, the depth map will naturally embed this information into the result.

For Fig. 9, the style of presentation is the same as Fig. 8. Figs. 9a-9c illustrates an outdoor scene, that is similar what was seen of Boston. The depth map is a very good representation of how an individual would perceive the scene to be. The last two examples show conversion of cartoons, demonstrating that our framework is not simply limited to just scenes captured by a camera. Figs. 9d-9f and Figs. 9g-9i show a frame from the Naruto Japanese Anime, and from Superman: The Animated Series, both share that they require very few strokes to obtain a stereoscopic images for good depth perception.

Finally, Fig. 10 shows examples using artwork. Specifically, Fig. 10a is a picture of Bernardo Bellotto’s Florence, and Fig. 10d is a picture of Thomas Kinkade’s Venice. With these images, they required more
strokes to obtain good depth perception, as there are many soft edges around the objects. As depth estimation is considered as a multi-label segmentation, objects that are classified as a certain depth may have their surroundings classified as the same depth due to their edges blending into the background. As such, extra strokes are required to be placed around the objects to minimize this “leaking”. Nevertheless, the depth maps that are generated show a good estimation of how a human would perceive the depth in the scene.

B. Results: Videos

In this section, we demonstrate conversion results on monocular videos from a variety of sources. To provide a benchmark with our work, we also compare with our own implementation of Guttmann et al.’s framework. To ensure proper compatibility, Guttmann et al.’s framework only marks the first and last frames of the video sequence. We maintain the fact that all video frames in the sequence need to be marked to ensure the highest quality possible, and so our framework will demonstrate results using this fact. In order to be consistent, the depth map for the last frame created by the framework in [12] is created using the labels for the last frame generated by our label tracking framework, while the first frame is user-defined as required. Certain scenes require a different label tracking algorithm. For those scenes where the objects move perpendicular to the camera axis, the modified TLD tracker is employed, whereas the optical flow method is used for scenes only exhibiting horizontal motion. The result of the system generates a set of depth maps - one per frame - to be used for rendering in stereoscopic hardware.

We illustrate four examples of video shots, and the next section (Section III-D) illustrates subjective results, showing some conversion results to test subjects. We begin our results by using a test set available on the Middlebury Optical Flow database\(^3\), known as “army”. Though this sequence is small, it illustrates our point that the proposed framework generates better stereoscopic views in comparison to [12]. The edges and objects are well defined, and if a conversion framework does not perform well here, the framework will most likely not work with other video sequences. In addition, this sequence is a good candidate for using the optical flow method, as the objects only exhibit horizontal motion. As such, Fig. 11 illustrates a conversion result using this dataset. Figs. 11a and 11b show the original first and last frames of the sequence, while Figs. 11c and 11d show their user-defined labels. Figs. 11e and 11f illustrate a sample depth maps in the middle of the video sequence, using the framework in [12], and the proposed method respectively. Finally, Figs. 11g and 11h illustrates anaglyph images, presented in

\(^3\)http://vision.middlebury.edu/flow/
the same order as Figs. 11e and 11f. As seen previously, these require red/cyan anaglyph glasses to view the results.

As can be seen, the depth map that Guttmann et al.’s framework generated does not agree perceptually in comparison to the proposed framework. There are various black “spots”, most likely due to the misclassification of the anchor points performed in the SVM stage. There is also noticeable streaking, most likely because the edges are one of the factors used when solving for the depths. If the definition of the edges are improper, then the results will also be poor. Though some areas of the depth map using [12] are well defined, other areas tend to “bleed” into neighboring areas, meaning that their definition of edges is not sufficient enough to capture these edges. Our framework not only respects object edges, but allows some internal variation within the objects to minimize the cardboard cutout effect. Fig. 12 illustrates a conversion using the Shell-Ferrari-Partizan (SFP) sequence. This video shot is of particular interest, as the race car begins in the shot to be very close to the camera, and eventually moves away towards the end. This scene is a good candidate for using our motion-based label tracking approach. As the depths should dynamically adjust themselves for the object as it moves away from the camera. The style of presentation is the same as Fig. 11.

Again, there are misclassifications or “spots” seen in the depth maps by Guttmann et al.’s framework, and noticeable streaking in areas where there are no strong edges. Our framework respects strong edges, and the depths of the car dynamically change as it moves away from the camera. We show two more results in Figs. 13 and 14. Fig. 13 is a clip from the previously seen “Sintel” sequence, but in this instance, Guttmann et al.’s framework will be used to compare with our framework. Also, the sequence exhibits horizontal motion, and no objects are moving perpendicular to the camera axis - a good candidate for the optical flow based label tracking method that we propose. Fig. 14 is a clip from the Superman: The Animated Series cartoon, showing a flying ship starting from the top, and moving towards the bottom, getting closer to the viewer - again, a good candidate for our motion-based label tracking method. In addition, to show the flexibility of our system, we use a colored label scheme, much like the one in Section II-B. As seen, the same observations can be said regarding respecting the edges of objects, and the dynamic adjustment of the depths.

C. Execution Time

The overall time taken to convert the images is dependent on the complexity of the scene, the size of the image, and the number of user-defined labels placed. On average, it takes between 30 seconds to 1 minute for labeling, while roughly a couple of seconds to generate the depth map. These experiments
were all performed on an Intel Quad 2 Core Q6600 2.4 GHz system, with 8 GB of RAM. After, should the user be unsatisfied with the results, they simply have to observe where the errors are, place more user-defined labels, and the algorithm can be re-run. For video however, the computational complexity has significantly increased, thus resulting in an increase in computational time. Also, it is heavily dependent on the number of frames. For Guttmann et al.’s framework, the time required for the SVM training stage depended on how many user-defined labels there existed. The more labels introduced, the more SVMs
Fig. 13: Sintel Dragon. First and last frames, and labels overlaid. Second row - Depth maps using the framework in [12] and the proposed, with their anaglyph images.

Fig. 14: Superman. First and last frames, and labels overlaid. Second row - Depth maps using the framework in [12] and the proposed, with their anaglyph images.

that are required for training. For our results, we used between 7 to 10 unique user-defined labels per keyframe, and training took between 75 to 90 seconds on average. To complete the depth map estimation, creating the sparse system took 10 to 20 seconds on average. Solving the actual system varied from 75 to 85 seconds. For our proposed framework, the depth maps were obtained in roughly 45 to 55 seconds. However, the label tracking is the more computationally intensive step, and took between 35 to 45 seconds on average. Even with this step, our framework is faster than Guttmann et al.’s framework.
D. Results: User Poll

As a supplement to the previous section, performance cannot be measured unless subjective tests are taken, as the overall goal is to create content for stereoscopic perception. We gathered 10 graduate students, and 25 undergraduate students with different acuities and viewpoints on their preference on viewing 3D content. The range of ages from these students is from 20 to 30. The viewing medium was a PC, equipped with an Alienware 1080p AW2310 23” stereoscopic 3D monitor, using the nVidia 3D Vision Kit. In addition to the results in Section III, we generated stereoscopic content from other sources. Each student was instructed on how the test was to be performed: (1) The student sat roughly 1 metre away from the screen. (2) The student was told that 11 video shots were to be presented to them stereoscopically. (3) For each shot, the student was shown one version of the shot, and the other version immediately after. One version of the shots was converted using the proposed method, while the other was with Guttmann et al.’s method. The order of presentation was randomized per shot to ensure unbiased results. (4) The student votes on which of the stereoscopic shots they preferred. Table II shows these subjective results. As seen, an overwhelming majority of students preferred our method, most likely because the depth maps generated in our work are more complete. They respect edges, and propagate the depths properly throughout the scene. With respect [12], a good amount of points were classified as having no depth, and so no depth perception could be properly perceived. However, there were a few that did prefer this content more, most likely because they were not receptive to 3D content in the beginning, and preferred the monoscopic version, which are essentially the depth maps generated by [12]. The students preferred the cartoon footage more with our method, most likely because cartoons have crisp edges, naturally favored in our method. What performs the “worst” are the television shows, most likely as there was a reasonable amount of motion blur, as we chose shots exhibiting high amounts of motion. This is understandable, as it is common for viewers of stereoscopic content to experience visual discomfort with high amounts of motion, as the disparities can widely change throughout the shot.

IV. Conclusion

We presented a semi-automatic system for obtaining depth maps for unconstrained images and video sequences, for the purpose of stereoscopic 3D conversion. With minimal effort, good quality stereoscopic content is generated. Our work is similar to Guttmann et al., but our results are more pleasing to the viewer, better suited for stereoscopic viewing. The core of our system incorporates two existing semi-automatic image segmentation algorithms in a novel way to produce stereoscopic image pairs. The incorporation of Graph Cuts into the Random Walks framework produces a result that is better than either on its own. A
<table>
<thead>
<tr>
<th>Movie Shot</th>
<th>Type</th>
<th>% for proposed</th>
<th>% for [12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avatar</td>
<td>Motion Picture</td>
<td>82.8%</td>
<td>17.2%</td>
</tr>
<tr>
<td>Superman</td>
<td>Cartoon</td>
<td>91.4%</td>
<td>8.6%</td>
</tr>
<tr>
<td>Naruto</td>
<td>Cartoon</td>
<td>88.6%</td>
<td>13.4%</td>
</tr>
<tr>
<td>The Pacific</td>
<td>Television Show</td>
<td>62.9%</td>
<td>37.1%</td>
</tr>
<tr>
<td>Star Trek: The Next Generation</td>
<td>Television Show</td>
<td>71.4%</td>
<td>28.6%</td>
</tr>
<tr>
<td>Open Season</td>
<td>CGI</td>
<td>88.6%</td>
<td>13.4%</td>
</tr>
<tr>
<td>Big Buck Bunny</td>
<td>CGI</td>
<td>91.4%</td>
<td>8.6%</td>
</tr>
<tr>
<td>Sintel</td>
<td>CGI</td>
<td>82.8%</td>
<td>17.2%</td>
</tr>
<tr>
<td>24</td>
<td>Television Show</td>
<td>77.1%</td>
<td>28.9%</td>
</tr>
<tr>
<td>Planet Earth</td>
<td>Nature Documentary</td>
<td>80%</td>
<td>20%</td>
</tr>
</tbody>
</table>

TABLE II: Table illustrating subjective results recorded by 35 students, using a variety of different shots from different media.

A modified version that ensures temporal consistency was created for creating stereoscopic video sequences. We allow the user to manually mark keyframes, but also provide two computer vision based methods to track labels for use, depending on the type of scene at hand. This alleviates much user input, as only the first frame needs to be marked.

However, the quality of the final depth maps is dependent on the user input, and thus the depth prior. With this, we introduced $\alpha$ to control the depth prior contribution, mitigating some of the less favorable effects. For future research, we are currently investigating how to properly set this constant, as it is currently static and selected a priori. We are investigating possible means for adaptively changing $\alpha$ based on some confidence measure to determine whether one paradigm is preferred over the other.

For video, the labeling process is quite intuitive. Once the user has provided the labeled keyframes, if there are errors, they can simply augment or adjust the labels until the correct depth perception is acquired. One avenue of research is to use Active Frame Selection [31], where a prediction model determines the best keyframes in the video sequence for the user-defined labeling. Once determined, any user labeling that result with these frames generates the best segmentation, and the best possible stereoscopic content. Another area of research is to discard object tracking completely, and rely on the optical flow method, as it can be more efficiently calculated. However, there is no scheme to dynamically adjust the depths of areas and locations by only using the flow vectors. As such, some classification scheme, similar to the SVM stage seen in Guttmann et al.’s framework may work here. Overall, our framework is conceptually simpler than other related approaches, and it is able to achieve better results.
REFERENCES


Raymond Phan is currently a Ph.D. candidate with the Department of Electrical and Computer Engineering at Ryerson University in Toronto, Canada. He received his B.Eng. and M.A.Sc. degrees in 2006 and 2008, respectively from the same department and institution. He is currently a Natural Sciences and Engineering Research Council (NSERC) Vanier Canada Graduate scholar, and a Ryerson Gold Medal winner. His current research interests are in multimedia signal processing, image archival and retrieval, computer vision, machine learning, stereo vision, and 2D-to-3D conversion. He is a student member of the IEEE, actively involved with the IEEE Toronto Section, as well as academic and extra-curricular groups at Ryerson.

Dimitrios Androutsos received his B.A.Sc., M.A.Sc., and Ph.D. degrees from the University of Toronto, Canada in 1992, 1994, and 1999, respectively, all in electrical and computer engineering. He has held numerous industry positions in both the U.S. and Canada and currently is a full professor at Ryerson University, Canada in the Department of Electrical and Computer Engineering. His recent active research interests are in the areas of image and video processing, stereoscopy, 3-D digital cinema, 2D-to-3D conversion and HDR imaging. He is an IEEE Senior Member, was Associate Editor of IEEE Signal Processing Letters, and was a guest editor of IEEE Signal Processing Magazine.