Adding Depth to Classical Cartoon Animations

by

Jinchao Sun,

Thesis

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science

University of Dublin, Trinity College

September 2009

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Jinchao Sun

September 7, 2009

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Jinchao Sun

September 7, 2009

Acknowledgments

Acknowledgements I would like to thank Dr Daniel Sýkora for all his help, guidance and advice throughout the course of this project. To all my lectures who taught me so much during the last year. To all my classmates for working alongside me and giving me so much help. And most importantly I'd like to thank my parents for everything. Without them this project would not have been possible.

Jinchao Sun

University of Dublin, Trinity College
September 2009

Adding Depth to Classical Cartoon Animations

Jinchao Sun
University of Dublin, Trinity College, 2009

Supervisor: Daniel Sýkora

Abstract Theatres featuring stereoscopic projection such as IMAX become extremely popular in last decade and attract large audience. However, key limiting factor for such cinemas is expensive movie acquisition phase. Lack of depth information in classical film footage restricts offer to movies created by specialized multi-camera system or 3D computer animation.

In this project we focus on developing a system that allows artists to add artificial depth information to classical cartoon animations. We first utilizes LazyBrush, flexible painting tool proposed in [13] to segment the image interactively and efficiently. Then we allow user to add depth value to every single regions we retrieved from the first step. The normals of the objects in the scene are then approximated with the help of the segmentation and depth information.

With all the data we get from the previous steps, we are able to render the scene with realistic lighting and shading. By enhancing the image low-pass filtering the depth map, we are able to create stereoscopic 3D image that gives a very strong depth feel. And by applying proper shading to each region, we are able to get a final stereoscopic 3D effect that's realistic and rich however simple in the underlying representation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.1.1 Traditional Hand Drawn Animation vs CGI

So why do we care so much about traditional animation? Isn't the time for pencil and paper long gone? Aren't the leading studios like Pixar and Dreamworks all switched to computer generated 3D animation? Well, things might be a bit different than you think.

First, not everyone favors CGI. There are definitely a lot of great CG pieces made in recent years, like "Kung Fu Panda", "Wall-E", "Ice Age: Dawn of the Dinosaurs" or "Up". However, more and more people have started to notice the big hits created on the other side of the world. Miyazaki's "Ponyo" is a great proof of what had happened.

Even studios that have been making successful computer generated 3D animations started moving back to traditional animations. After switched to CGI for several years, Disney animation is now putting a stop on it and returning back to the traditional hand-drawn animation. After five years , Disney recently released their latest hand-drawn animation "The Princes and the Frog". And they have another one in production, the re-creation of award winning "Beauty and the Beast".

Who says hand-drawn feature cartoons are dead? They're just hibernating.

1.1.2 Stereoscopic 3D

3D movies have been attracting more and more attentions over the years. Due to the advantage that stereoscopic 3D movies encode depth information in the movie footage, it gives people a more realistic feeling of the scenes.

These days, a lot of movie theaters have been equipped with stereoscopic 3D projection system (e.g. IMAX, RealD) to allow audiences to view 3D movies.

With the fast growing technologies, the effect of stereoscopic 3D is becoming much better over the years. The old anaglyph has been replaced by the normal stereo pictures. LCD shutter glasses and

Polarized glasses have replaced the original red and blue glasses. Lately, the newly designed lenticular, barrier and light-field screens even enabled people to view 3D scenes without any glasses.

People are beginning to notice the huge market behind stereoscopic 3D movies. In 2005, the 3D version of Chicken Little earned about 2.5 times as much per screen as the flat version. So far this year, several stereoscopic 3D animations have been released, e.g. Coraline, Monster vs. Aliens, Up, Ice Age: Dawn of the Dinosaurs.

Figure 1.1 from Nvidia illustrates the popularity increase of stereoscopic in recent years.

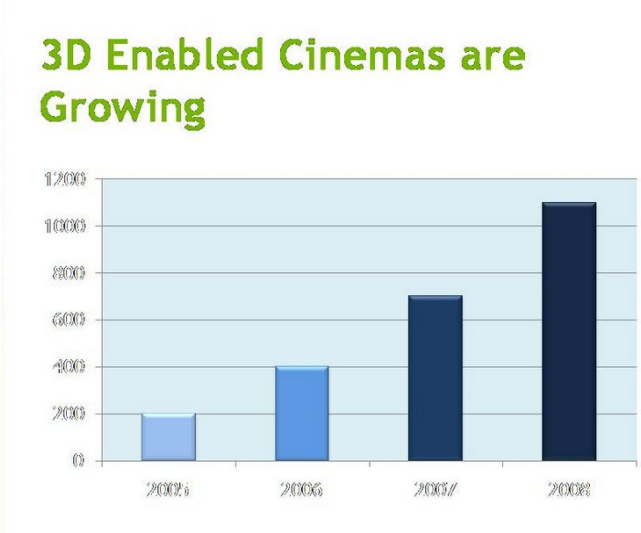


Figure 1.1: Contrast Enhancement Map Created from the Depth map.

Stereoscopic 3D technology is becoming more and more popular not only in the movie industry but also other fields. Games industry is another hot spot where stereoscopic 3D technology is pioneered. On August 6th 2009, the S-3D Gaming Alliance (S3DGA) was Announced during Siggraph. And on 27th of the same month, Electronic Arts developer joins S-3D Gaming Alliance(S3DGA) Advisory Board. Dimension 3, the international forum on 3D stereoscopic technology, now on its third edition, is experiencing rapid growth of the community.

1.2 Problems in Converting to Stereoscopic 3D

We all know that stereoscopic means that we need to have two separate images for every single frame. The stereoscopic 3D animations mentioned above are all created from a 3D scene. This process, compare to creating stereoscopic 3D from 2D animation is much easier. After the scene is set up, they render one frame from the current camera position. Then they move the camera and render another frame from a different angle. The actual process is obviously more complicated than I had described but that was the basic idea.

In comparison to that, converting the classical animations into stereoscopic 3D animation is a very

expensive process. There's no depth information about the objects or characters in the scene at all. Consequently, adding depth information into the original hand drawn animation becomes the number one task in converting hand-drawn 2D animation into stereoscopic 3D animation.

1.3 Different Approaches

Several methods have been developed to acquire depth information from the original film footage. For instance, structure from motion (SFM) can convert a scene (viewed from different angles) on film footage into 3d object if the scene is very close to reality (no exaggerations, no change of shapes). Using multi-camera system can capture scenes with depth information automatically but it doesn't suit our situation. If the original animation is in computer 3D graphics, we can simply introduce another camera into the original scene and take an extra capture each frame.

What if there is only a single view? Tour into the Picture [2] presented a method of adding a third dimension to a single image through polygons and planar regions. And similar to that, Sing Bing Kang et al. [5] [6] proposed similar methods of making a single picture multi-layered with more expressive depth variations and thus a more realistic rendering. Zhang Li [1] proposed a novel approach for reconstructing 3D scene models from a single painting or photograph. Given a sparse set of user-specified constraints on the local shape of the scene, a smooth 3D surface that satisfies the constraints is generated.

There are some other researches focused on SfS (Shapes from Shading). [14] described an interactive SfS method which efficiently uses human knowledge in order to resolve ambiguity.

However due to the fact that classical animations are made mainly by hand drawing, there is no way to acquire depth information with any of the techniques mentioned above.

There have been some novel algorithms proposed to infer depth information directly from 2D contours in the input image (without human interaction like the ones mentioned in the previous paragraph). Teddy [3] is such an interface for quickly and easily designing freeform models. The system constructs a 3D polygonal surface from a 2D silhouette drawing. It inflates the region surrounded by the silhouette to create shapes like stuffed animals. SmoothSketch, a more sophisticated algorithm was introduced later introduced [7]. This system not only works with simple closed curve as in Teddy, it analyzes the scribbles and infers the hidden contours and the topological shape.

Disney animation studio introduced a technique that can convert hand-drawn 2D animation into 3D models and then bend the original movie around that. In this way, they create stereoscopic 3D movies that's not just simply layers of flatness but a truly dimensional environment. However this technique may requires a lot more human interactions than the technique proposed in this project. Because generating 3D models from hand-drawn animations is a very complicated process and has various kinds of special cases which made the process very hard to automate. The detail of these techniques will be discussed in the next chapter.

1.4 Our Assumption

Before we started our project, we found that combing multi-layered approach with realistic rendering can achieve similar results of the smooth depth transition approach. By rendering each of the flat layers in a way that each one of them has a great 3D feel, the final results should be comparable with the smooth transition approach.

In this way, there is no need to use complicated algorithms to convert 2D drawings into 3D models. However, normals of the objects in the scene need to be calculated. Fortunately, extracting normal information from hand drawn images are much simpler than 3D reconstruction. In other words, using the multi-layered approach with extracted normal information for rendering can save a lot more time and energy than the approach to construct actual 3D models.

However, this is only our assumption, we need to continue the experiment and evaluate the result.

What we offer in this paper is a system that allows animators to add artificial depth information to a classical animation in an easy way. And the 2D hand drawing can then be rendered to a stereoscopic image using the depth information obtained during the previous process. The system is capable of handle the following tasks.

1. This system is able to separate the hand drawn image into different regions interactively. Imprecise actions (soft scribble) from the animators are also be picked up and compensated by the system.

2. The system allows animators to add/subtract depth to/from different regions in the original 2D scene.

3. Normal's are extracted from the original 2D image automatically. The 2D image can then be rendered with whatever illumination method the user choose to create photo-realistic or non-photo-realistic scenes.

4. Create two separate images for left and right eye respectively for future use.

In addition to that, the system implemented some techniques proposed in different papers to improve the final result. For instance, Luft et al proposed a technique that enhances the depth feel of the image by unsharp masking the depth buffer. Due to the limitation on 3D display devices, we can only use anaglyphs as our stereoscopic representation. However, with the necessary information provided, the system can be easily adjusted to render other types of stereoscopic images when more advanced devices are available.

Chapter 2

Related Work

2.1 Disney Animation Studio's Approach

During Siggraph, Disney animation studio presented their approach of converting hand-drawn animation into stereoscopic 3D (Talks: Capturing and Visualizing Animation Topic: Medial Axis Techniques for Stereoscopic Extraction - Stereoscopic conversion of Disney's "Beauty and the Beast" required development of novel extensions to standard medial axis techniques to automatically generate depth maps from the hand-drawn images. Given by: Even Goldberg, Walt Disney Animation Studio). They have been using this technique in the production of the stereoscopic remake version of the classical "Beauty and the Beast". This so called Medial Axis technique used for stereoscopic extraction is based on skeletonization/medial axis transform.

For audiences who are not familiar about skeletonization, it is a process for reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels. Here is an example in figure 2.1

Medial axis transform is often used interchangeably with skeletonization however slightly different. Skeleton, as we can see in figure 2.1, is a simply a binary image. Medial axis transform, on the other hand, is a gray-scale image where the intensity of each pixel on the skeleton represents distance between the boundary of the original shape and itself.

What Disney animation studio did was,

1. They first select a certain object and start eroding the object by the medial axis technique.
2. They keep doing the first step however using different erosion level settings. The selected object is eroded by different erosion levels and a set of eroded version of the original image is created.
3. As we all know in stereoscopic 3D, the displacement/parallax shift of a certain object is proportional to the depth of that object. They calculate the displacement value for each of the eroded version of the image by the depth value.

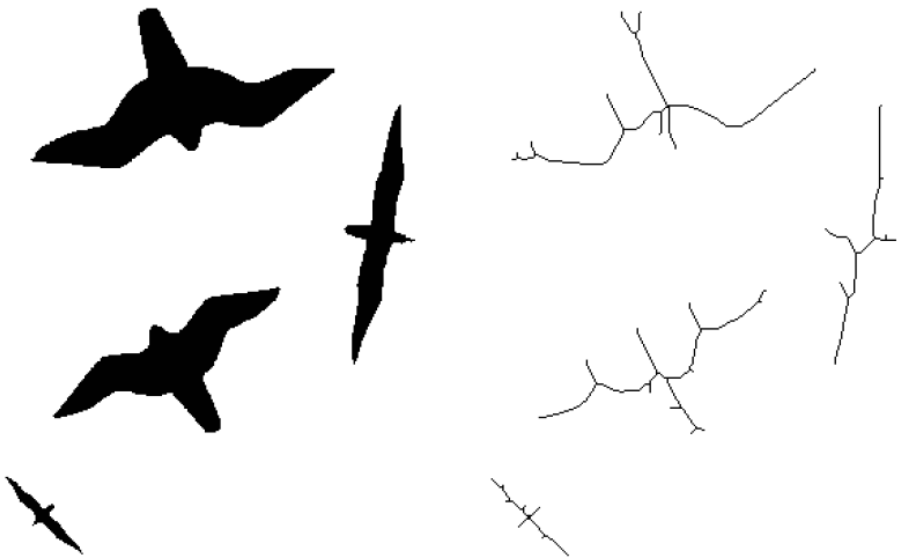


Figure 2.1: Skeletonization example, image from Computer Vision course note, Trinity College Dublin

4. Every one of the eroded version of the image is displaced by the displacement/parallax shift value calculated in the last step.

5. They blend the displaced images into one image and combine it with the original image to produce the final stereoscopic effect.

Imagine we have an image with four separate shapes on it (figure 2.2).

Figure 2.2: The original image with four individual shapes

We perform medial axis to this image using different erosion level settings and the results are listed in figure 2.3. The erosion levels are lower from left to right and top to bottom.

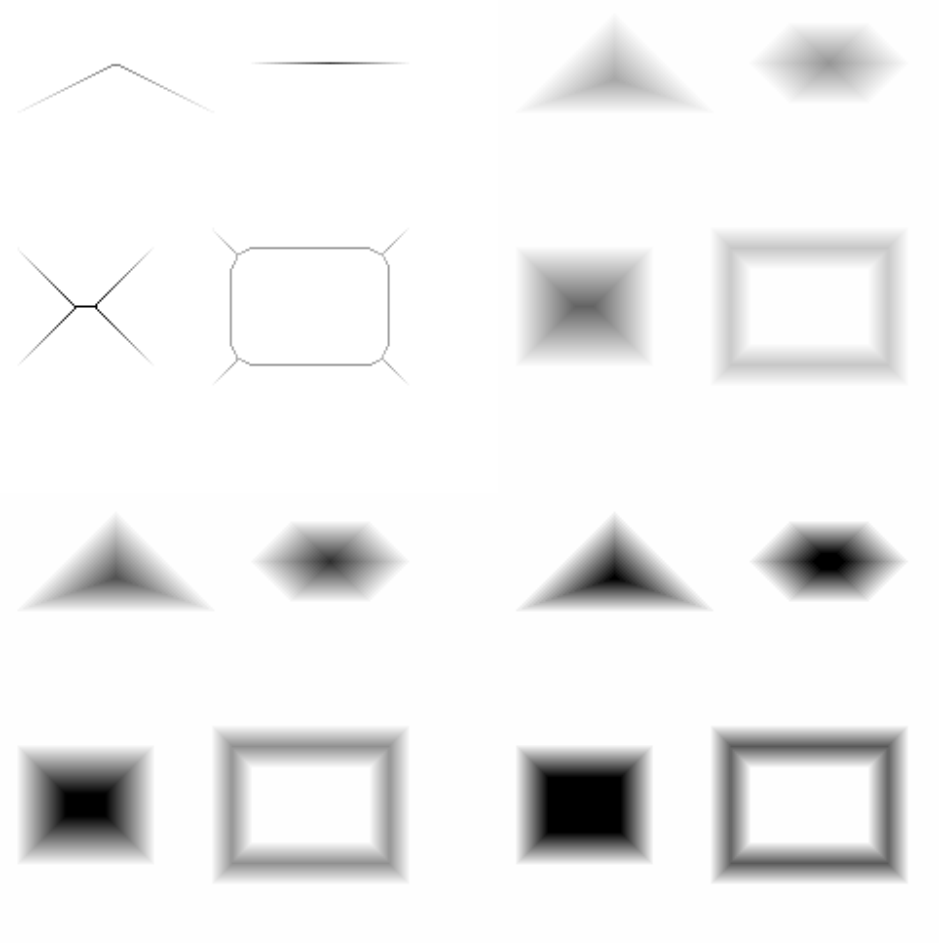


Figure 2.3: Medial Axis processed image with different erosion settings.

These are just 6 layers of the same image. In reality, Disney animation studio may have used much more layers to blend together and generate the final result. This approach is very simliar to volume rendering in some way. In volume rendering, the 3D data set is blended with a set of 2D slice images (normally acquired by a CT, MRI or MicroCT scanner). In the Disney approach, a single object is cut into many slices and blend back together after being displaced according to the depth value.

After retrieving layers at different depth of the same object, they displace the layers by a value that's related to the depth of that layer. If we look at the figure 2.4. The one on the left is the supposed object. The red arrow on the left represents the value by which the layer should be displaced.

This approach creates a truly dimensional environment rather than layers of flat comparing to our approach. However, in order to perform this technique, there are several requirements.

First, one of the reason why Disney chose "Beauty and the Beast" to create stereoscopic remake version was that the movie was done on a system called CAPS (Computer Animation Production System). In this system, the characters or objects are archived in separate layers and separate levels.

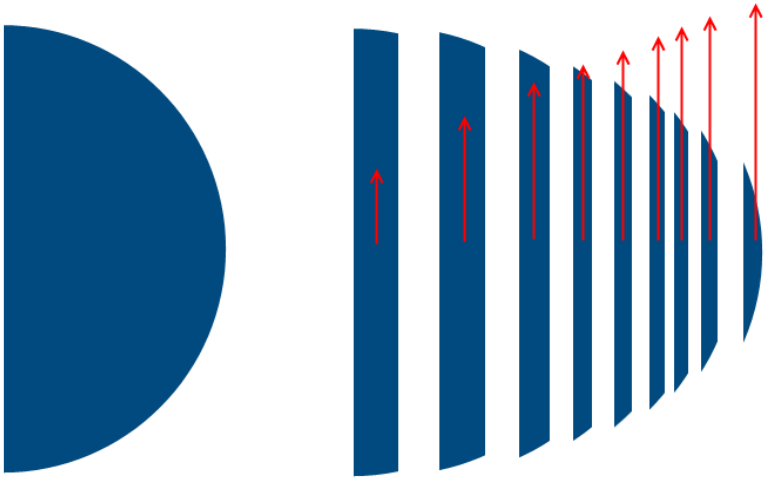


Figure 2.4: Side view of an object being cut up into pieces and displaced by different values

That's why when they went back to the system several years later, it is very easy to separate those layers into a depth of field and create the stereoscopic 3D effects. Unfortunately, a lot of the hand-drawn animations are not organized in this way.

Second, the skeletonization/medial axis transform is extremely sensitive to noises. Unfortunately noise is one of the most common things in hand-drawn animations. Here is an example (figure 2.5)

Our goal is the same as Disney, to create a stereoscopic 3D effect that the objects in the scene look like they have smooth depth transition rather than flat surfaces stacked on top of each other. However, we want the depth information to be fully controlled by the artists yet very easy to handle. To put it another word, we want to use the simple geometry representation to create realistic effect.

Think about bump mapping or normal mapping. In a game or animation project, objects are often represented by very simple polygonal mesh. When it goes into the rendering phase, a normal map is applied to the polygonal mesh in order to alter the original normals on the polygon surfaces so that lights will interact differently comparing to the previous situation.

This project has similar idea to bump mapping or normal mapping. In order to achieve better effect, rather than change the underlying geometries to be more complicated ones, we change the normals on the same geometries. To do this, we need to have a normal map calculated from the original geometry. How do we achieve that?

2.1.1 Lumo: Illumination for Cel Animation

This paper presented a technique to approximate lighting on 2D drawings. Using the technique proposed in this paper, the environment illumination can be easily applied to the hand-made drawings such that hand drawn animation can be integrated into live-action scenes. To illuminate a hand-made

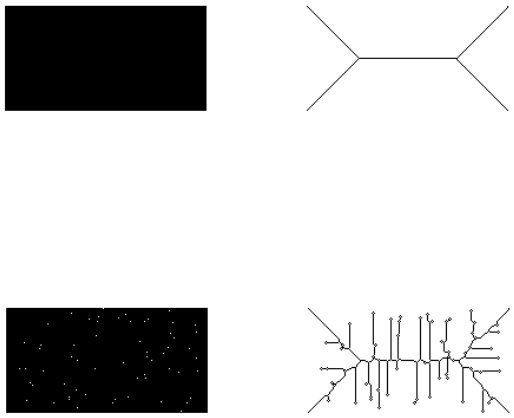


Figure 2.5: The noticeable result of medial axis when noises exist. Original image with no noise(top left); Medial axis of the original image(top right); Original image with noise(bottom left);Medial axis of the noisy image(bottom right)

drawing, there are two major problems:

A point on a surface requires information about position and the surface normal in order to be correctly illuminated. Unfortunately, in the hand drawn images, there is no information about surface normals and the position of a point is only in 2D space.

There are some previous work that are focused on estimating the geometry from the hand drawing. Igarashi et al developed a system called Teddy that can allow user to create 3D geometry just by hand scribbles. The system takes the image and analyzes it and then inflating the shape automatically to create the final 3D geometry. SmoothSketch, a more sophisticated system that converts more complex sketches to 3D free-form shapes was developed by Karpenko et al in 2006. More detail on these approaches later.

However, in hand drawn animations, the shape of the objects often change dramatically which would cause the 3D geometry to be created very differently. What's proposed in Lumo is a technique that utilizes the image to approximate the surface normals without having to get the 3D geometry first. The depth information are not very important to Lumo since there's no need for the viewer to look around the object.

The implementation of this method estimates normals wherever possible and uses sparse interpolation to approximate them over the remaining image.

One of the key idea behind the technique is that, in a orthogonal projected scene, the normals on the outline of the object are always perpendicular to the eye vector. In other words, the normals on the outline all have z components as zero in a right-handed coordinate system. Following this principle, several separate techniques are developed. The results generated from these different techniques are

blended together to produce the final result.

Blobbing technique: Normals can be extracted from the external boundaries of the object in the drawing. This technique works well when the shape is very simple. e.g. a circle in hand-drawing normally represents a sphere in 3D space. We can take the circle and work out the matte (a circle filled in with the same color as boundary). After doing a gradient calculation across the whole image, we will get normals on the boundary. And then the normals can be interpolated and create a normal map.

The basic blobby shape created from the exterior boundaries of a more complex image has very little details. To improve this, the paper suggested that regions should be separated first before applying the blobbing technique. However, although improved, the result still lacks the detail of the original image.

Quilting technique: To extract normal information from the lines that are not region edges, a pair of normals are generated on each side of those lines.The pair of normals points away from each other. However, the quilting technique produces artificial effects because for the lines that are not region edges, only one side of the edge should produce normal information.

Over/Under technique: For every boundary, there is a spatial relationship between the two side of the boundary. In other words, where there is a boundary, one side of it must be higher than the other side. In traditional ink and paint, artists would represent the inside of the regions using light ink lines along the edge and vice versa. In this paper, they suggested that user should perform an over/under process as an additional step.

However, they also suggested that for line segments that don't border paint regions, we can analysis the shape of the curve and assume that the region inside of a "C" curve is always on top of the region outside of it. This approach provide additional help to allow artist.

Confidence Mattes: The over and under technique gives us an image where every boundary is formed up with two colors. White color represents the "over" region and Black represents the "under" region. After this process, an interpolation should be performed and a grayscale matte is formed. This matte image not only represents the over and under relationship between different sides of edges but also can be viewed as a measure of confidence in the known normals.

The normal on the under side of the edge should be interpolated across the whole region of the underlying object as if the edge didn't exit. The normal on the over side should interpolate from the edge. Using this confidence matte, a normal map can be retrieved by blending the quilted line-based normals with region-based normals.

Sparse Interpolation: The paper suggested an approach to view the normal image as a damped spring. In this way, normals can be interpolated iteratively and the iteration should stop when the nodes on the "spring" are moving within a considerable small area. The Nx and Ny components of the normal are interpolated independently and the Nz component is calculated afterwards to maintain the unit length of the final vector. Detail implementation of this technique will be discussed in the implementation chapter.

Here is an example illustrating the process of Lumo. The images from left to right on the top row are: The blobbing technique applied only to the outline; The blobbing technique applied to each

regions; Quilting technique; Blended normal map; The images on the bottom row are: The scene rendered with a diffuse spheremap; The object with its basic color and blended into a real-life scene; The object scaled by illumination; The final scene.

Figure 2.6: The process of Lumo introduced in [4]

2.2 2D drawings to 3D

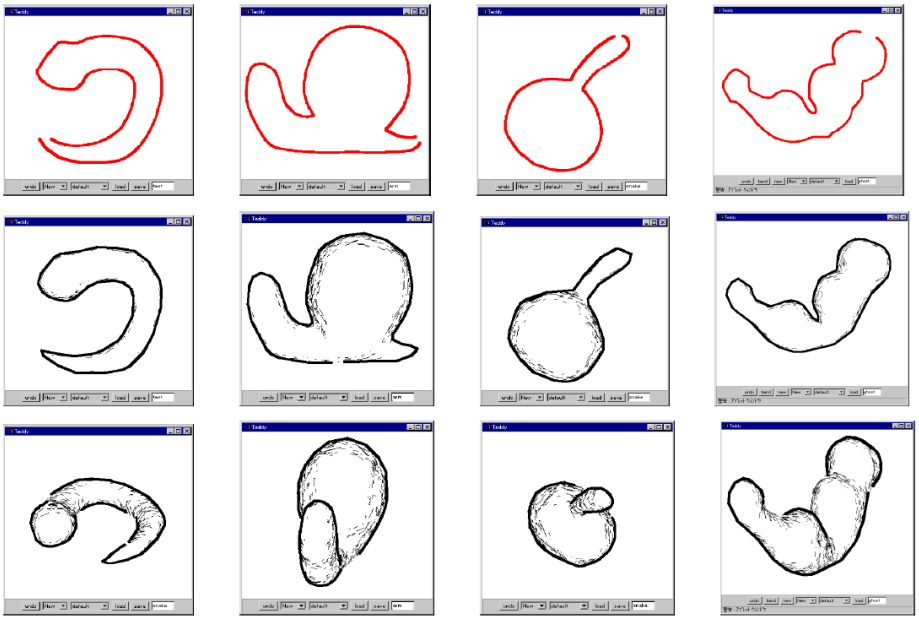
To convert 2D hand drawings to 3D, there are several approaches available out there.

2.2.1 Teddy: A Sketching Interface for 3D Freeform Design

Teddy, a interface application developed by Igarashi et al, is capable of converting 2D freeform strokes drawn by user into freeform 3D models. The results generated from user generated strokes are converted interactively. Although the final results are all models that look like stuffed animals or other rotund objects due to algorithm that's employed in the system, the results are quite plausible.The following image 2.7 illustrates some of the examples of using teddy to create freeform 3D models.

As we all know, 3D models are commonly represented as 3D polygonal meshes. Each stroke in Teddy would alter part of the 3D polygonal mesh hence produces alternative result. The main algorithm behind the system is divided into several steps.

1. In order to get rid of the noises that come with hand drawings, noises are inevitable whenever real life meets machine process, the system used the technique proposed in [9]. Every stroke from user is processed and converted to a smooth polyline.



a) snake b) snail c) cherry d) muscular arm

Figure 2.7: Examples from [3]

2. By connecting the start and the end point of the strokes, a closed polygon is generated. All the points on this polygon are in the same plane.

3. Using a algorithm called Constrained Delaunay Triangulation, the planar polygon is divided into multiple co-planar triangles.

4. Following the chordal axis approach introduced in [10], spines of the polygon are extracted. Part of the process of how this is achieved is illustrated in Figure 2.8

5. Then they elevate the vertices of the spine by values that are proportional to distance between the vertex and the polygon

6. The last step is to construct the final polygon mesh that wraps around the planar polygon and the elevated spine. These last two steps are illustrated in Figure 2.9

There are off course some other techniques introduced in this paper such as extrusion, cutting and smoothing. However they are only effective in the teddy system where the user input stokes are added to the image interactively. In our application, the hand drawings are scanned into the system so that those techniques are irrelevant to our application.

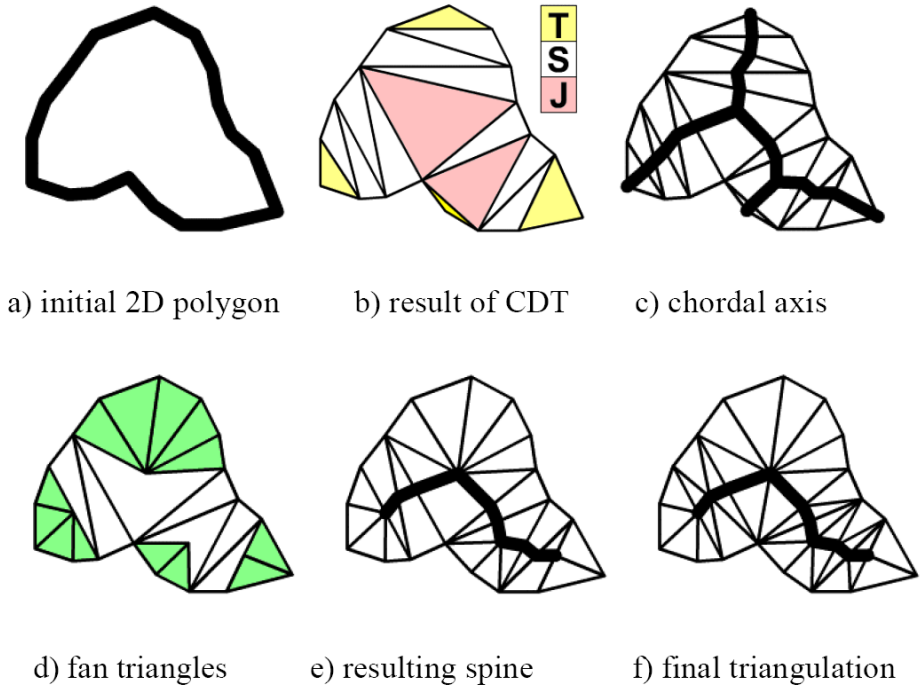


Figure 2.8: How Teddy performs triangulation and finds spine. Examples from [3]

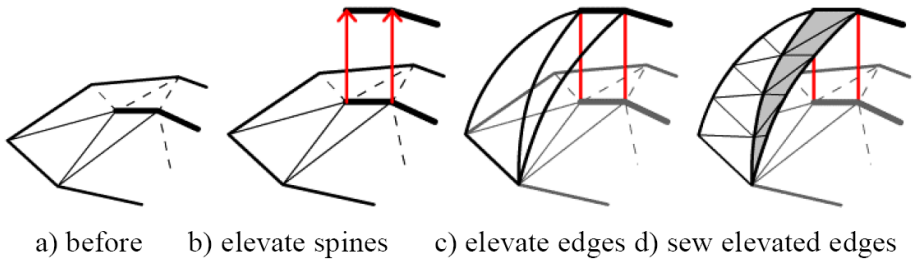


Figure 2.9: Elevating the spine and constructing polygon mesh. Examples from [3]

2.2.2 SmoothSketch: 3D free-form shapes from complex sketches

Teddy's inflation algorithm has pioneered this area and more algorithms have been developed to improve the results. In Teddy, one of the limitation is that only simple closed curve contours are considered as legal inputs. SmoothSketch [7], on the other hand, extended Teddy's work and allows user to draw more complicated shapes. An example is given in figure 2.10.

The algorithm that's used in SmoothSketch is as follows:

1. Figural Completion. User draws the visible contours. The system tries to find all the tees and

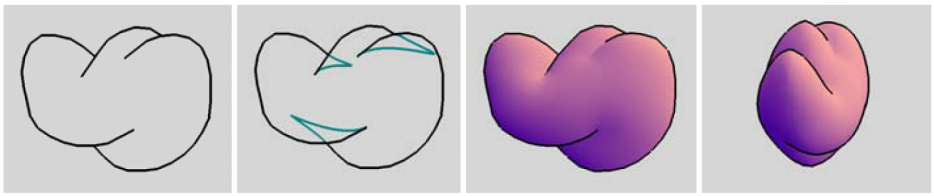


Figure 2.10: Example from [7]

cusps on the contours. The hidden contours are automatically completed and the system assigns Huffman's labels to them.

2. Paneling Construction. When all the labels are valid, the system creates panels from the labeled drawings completed in the previous step. Then it triangulates the panels, creates 3D copies and finds silhouette correspondences for the 3D panels.

3. Smooth Embedding. If there's no tees or cusps left, the system can process along. However, if it does, extra 3D vertices are added and depths are assigned to those 3D vertices using a mass-spring system. The system assigns depths to edges and panel interiors in order to get rid of all the tees and cusps. After all of the tees and cusps are eliminated, the panels are stitched into a mesh. The system then inflate the mesh using a mass-spring system to create the final shape. A Taubin's smoothing is optionally applied to produce better results.

2.3 Image Segmentation

Image Segmentation is one of the foundation stones for this project. In order to vary the depth of different objects in the scene, the system has to distinguish between the regions in the image. There hasn't been a lot of image segmentation method specially designed for depth information generation. However, in the field of coloring or painting hand drawn images, a lot of work have been done.

One of the most suitable technique is developed by Sykora et al [13] in 2009. This paper presented a novel interactive way of painting hand-made cartoon drawings and animations. Comparing to some previous works, the LazyBrush system has better simplicity and flexibility. LazyBrush does not rely on style specific features such as homogenous regions or pattern continuity yet still offers comparable or even less manual effort for a wide range of different drawing styles.

Here (figure 2.11) is an example of image segmentation using LazyBrush technique. The original hand-drawn image and user color scribbles are passed into the system and it outputs an image with different regions colored according to the user input.

Painting, or adding colors to hand drawings, is a common functionality provided in most of the image creation/manipulation tools. For example, Microsoft Paint, Adobe Photoshop, Paint .NET, etc, they all have a tool called Paint Bucket. However, in almost all of the tools, a flood-fill algorithm is employed in the implementation of the painting functionality. The flood-fill algorithm works well

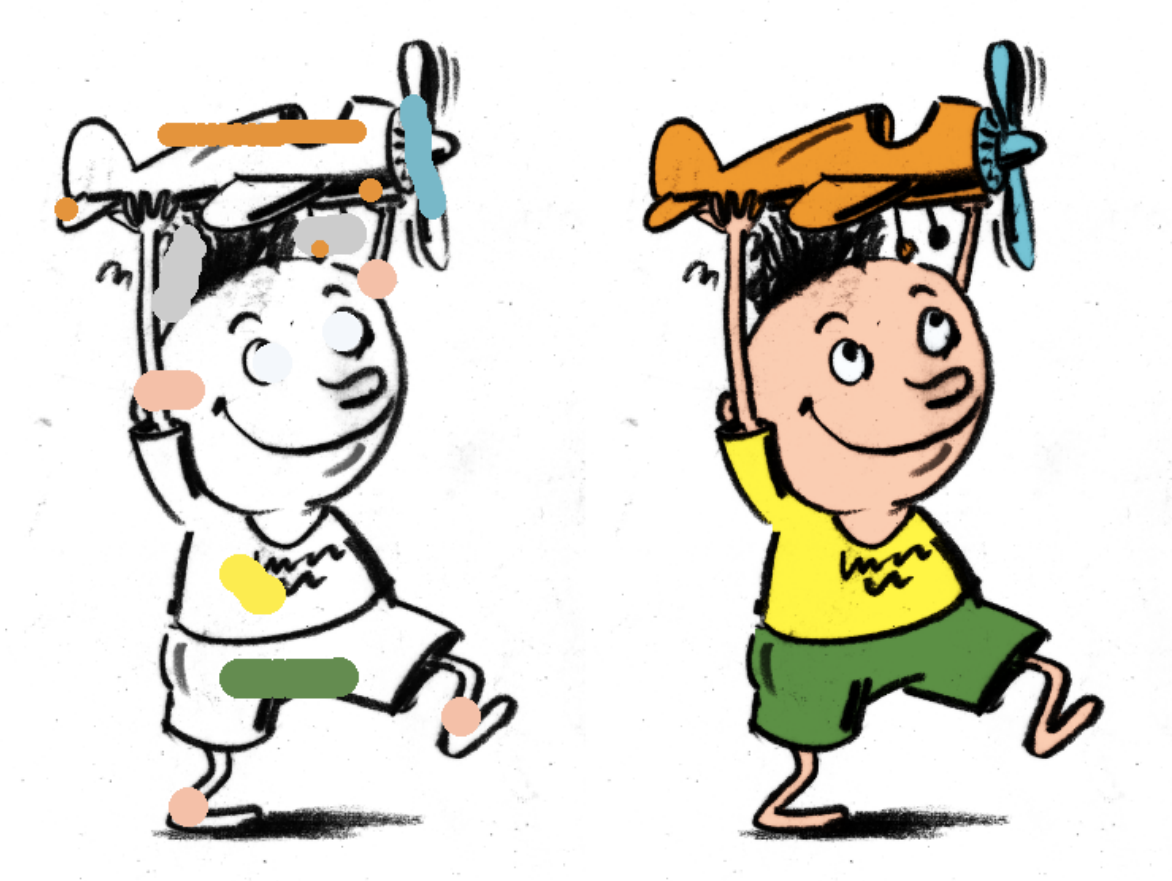


Figure 2.11: Using LazyBrush to color the hand drawn image. Example from: [13]

with for images with "perfect" drawings that have salient continuous outlines. And if the drawings have a piece of homogeneous region that is covered by disturbing textures, the flood-fill algorithm could fail easily.

Unfortunately, in hand-made drawings, the problematic features like gappy outlines and disturbing textures separating homogeneous regions appear quite often. In order to paint hand-made drawings, artists often have to do some preparation work such as connecting the gaps on the outlines. Even so, the painting process could still be very tedious. When they use paint bucket to paint a homogeneous region that's separated by textures, a lot of manual work have to be put into it. Even worse, imagine an animation that have thousands of frames where each frame must be painted, the amount of work has to be done is huge and the process is extremely time consuming.

In manga colorization [11], the algorithm calculates the image-based likelihood such as pattern or intensity similarity. They assume that there is a on-to-one correspondence between color and image patterns or intensities. However, in typical hand-drawn animations, repetitive patterns or intensity variations are not typical hence the result can be very poor.

2.4 Rendering

One of the main goals of this project is that we want to display richer stereoscopic 3D results with simple underlying structure. To achieve this, we choose to render each of the reasons such that they would appear to be more complex. For instance, in our case, we render flat surface in a way that they appear to be blobby.

We can obtain a normal map from the first couple of steps and when it goes into the rendering stage, we want to do it in some way so that we can get better result without adding more polygons to the original polygonal mesh.

2.4.1 Normal Mapping

Normal mapping, a technique for bump mapping is used to add details to geometry without adding more polygons. In SIGGRAPH 1996, Krishnamurthy and Levoy introduced this new idea of taking geometric details from a high polygon model and create a displacement map over nurbs. It is an ideal choice for our application. In our blinn-phong shading model, a halfway vector (the vector in the middle of camera and light source vectors) is calculated and dotted with the normal. This normal can be replaced by the normal we get from the normal map. In this way, the lights will have a different effect on the original geometry although the geometry hasn't been changed.

The normal mapping technique and the blinn-phong shading model are all 3D rendering techniques. We made some adjustment to make it working in our situation. We approximate the view vector to be always pointed at (0, 0, 1) which is out of screen. And because we don't have any vertices information, the vertices are approximated from the texture coordinate of the image. Everything is performed as a post effect in a pixel shader.

2.4.2 Sphere Mapping

As a special rendering techinique, sphere mapping is proposed in the [4] paper. The key idea behind this it that a unit sphere has the special property that any point on the sphere corresponds exactly to its surface normal at that point.

Following this idea, to render a scene, we only need to look at the normal map and use the sphere map as a lookup table. We retrieve the color value from the sphere map where the normal vector is in the normal map is equal or very close to the same point on the sphere map.

This approach allows user to change the art style of the rendered scene quickly and easily. Due to the fact that the lighting and shading are not calculated but rather looked up on a unit sphere, users can easily change the art style by changing just the rendered unit sphere. In addition to that, the unit sphere can be pre-rendered using a professional render tool so that it will provide a better and more detailed rendering which can be used to achieve better final effect.

Here is an example of sphere mapping (figure 2.12). The teapot on the left is rendered in a cel animation style by using a cel animation style rendered unit sphere as a lookup table. Same applies to the teapot on the right.

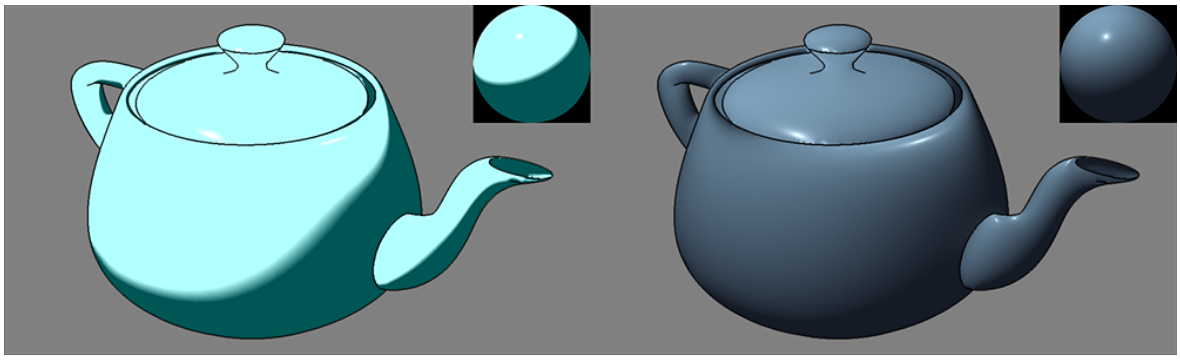


Figure 2.12: Example of sphere mapping. Example from [4]

2.4.3 Image Enhancement by Unsharp Masking the Depth Buffer

Given a single image, how can we improve its realistic 3D effect perceptually? A simple but effective technique developed by Luft et al. [8] is a very good choice. The method they proposed in this paper was originally from a technique that painters have been using for hundreds of years. Painters tend to improve the perception of objects in an image by enhancing contrast in a local region. As you can see in the example paintings/drawings in figure 2.13, the contrast between the boundary of the objects and the background scenes are exaggerated in order to achieve better sense of space.

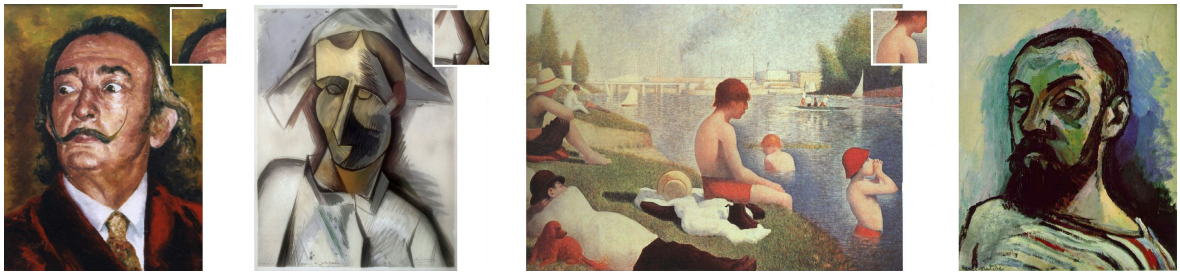


Figure 2.13: Example of classic paintings and drawings using contrast enhancement. Image from [8]

In a scene where the spatial arrangements of objects within are very complex, standard shading technique can often generate scenes that have qualities that couldn't meet users expectations. The result of this dull appearance is especially noticeable in the shadowed area or area that is strongly affected by ambient light.

The pixels that we're interested in are the ones surrounding the objects in the foreground. To retrieve these pixels, a mask need to be calculated. How to mask out the ones that are irrelevant? The figure 2.14 illustrates the main process involved in the technique.

Although the technique proposed in this paper was inspired from paintings, it is transferrable to photo-realistic rendering. The example photo-realistic rendering of the scene in the figure 2.15 shows that.

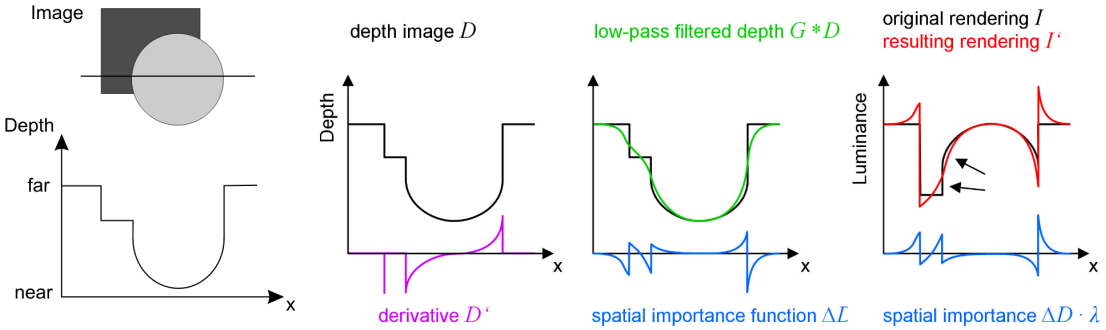


Figure 2.14: How is the mask calculated from the depth map. The original image and its related depth map on the line (first); The rate of change of the depth (second); The filtered depth and the spatial importance function (third); The resulting luminance with certain areas enhanced (last). Image from: [8]

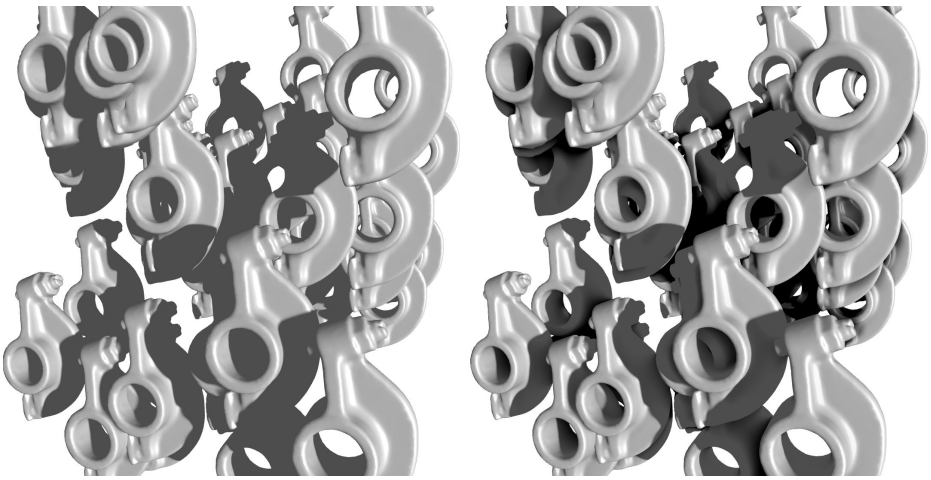


Figure 2.15: Results of the Image Enhancement technique: original rendering (left), enhanced rendering (right). Image from:[8]

Chapter 3

System Design

3.1 Representation of the Stereoscopic Results

As can be seen in figure 3.1, one approach is to divide the original image into multiple layers and assign different depth value to each one of them. The other one is a more realistic way of adding depth information however more complicated.

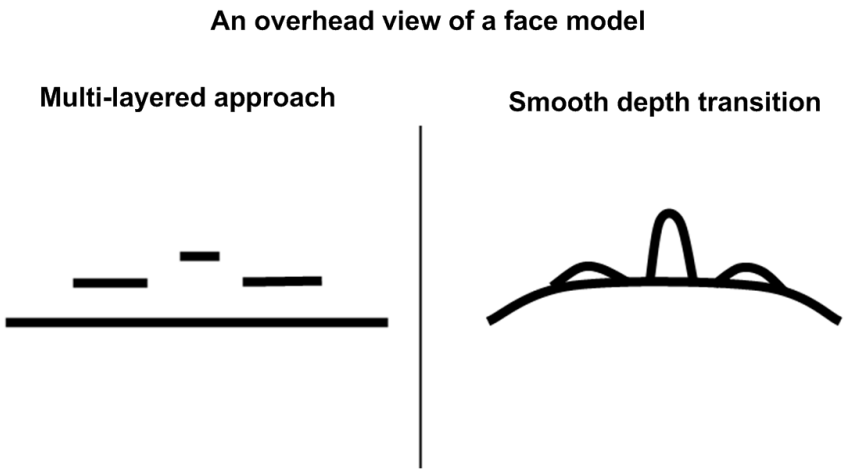


Figure 3.1: The two types of representation of the stereoscopic results

The question may sounds a bit strange. You may wonder why we propose this question, "the second approach is obviously much better than the first one". Strangely enough, that's not the case.

Research shows that when the internal volume (roundness) is limited, human wouldn't notice the difference between the two approaches below. As shown in figure 3.2, the stereo separation quickly approaches maximum if the object is very distant to the viewer. That's why in the stereoscopic 3D movie industry, the depth information of objects are usually exaggerated.

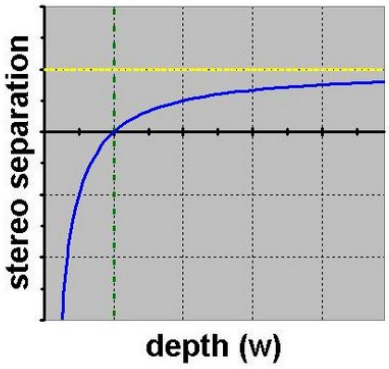


Figure 3.2: The relationship between depth(w) and and stereo separation. Image is from Nvidia. Stereo separation is the distance between left and right eye images of the same object. The maximum value for stereo separation is equal to the distance between the pupils of a persons' eyes which is about 2.5 inches.

What if we use the multi-layered approach and render each layer in a way that it looks more 3D? Following this assumption, the system took the direction of the multi-layered approach.

Interestingly enough, during Siggraph this year (2009, in New Orleans), I talked to the stereoscopic director of Dreamworks studio Phil Captain3D McNally and he mentioned something very interesting. After the production of "Kung Fu Panda" (Non-stereoscopic 3D Animation film that was nominated for Oscar and had 11 wins and 20 nominations for other awards), they did a in house stereoscopic effect test and convert a scene of two minutes into stereoscopic 3D.

The whole scene was converted in a way that all objects in the scene are separate layers without any smooth depth transition. However, to their surprise, the result effect of that scene was "fantastic". Our original assumption was well proved.

3.2 The System Process Design

In order to achieve the goals mentioned in the introduction chapter, we designed our system like this (Figure 3.3).

The user would scribble on an interface that's provided as part of the system. And it would be combined with the input image and sent to segmentation analyzer.

The segmentation analyzer (which is based on LazyBrush) calculates the best solution for segmentation according to both the user inputs and the original image. Note that the segmentation analyzing is an iterative process such that it runs every time when user makes an input.

When the user is happy about the result, the segmentation analyzer would give all the information about segmentations and the depth of them to the renderer. The renderer will render the final image with depth information encoded in it.

In the LazyBrush approach, regions are being separated while user input color scribbles. However,

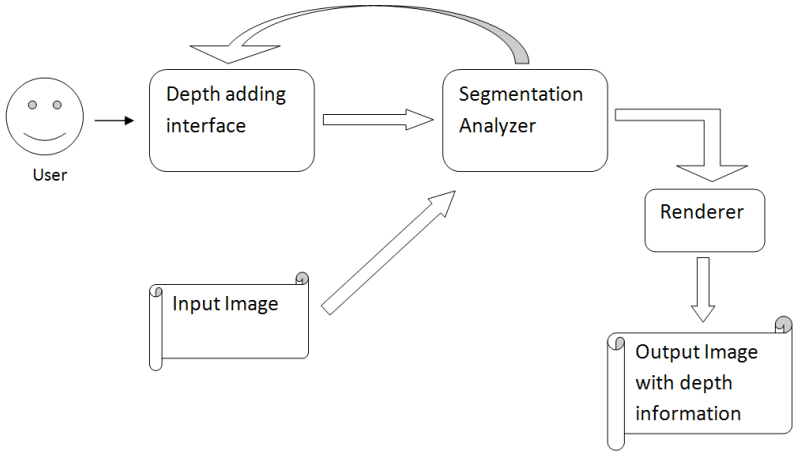


Figure 3.3: The original design of the process

Figure 3.4: User inputs depth scribbles

due to the difference between the representation of color and depth, the same approach wouldn't work as well as in LazyBrush. Imagine when the user want to separate two regions while assigning different depth to them, in order to use scribbles like in LazyBrush, they need to draw something like figure 3.4.

The original designed didn't count in the fact that depth information is not very easy to be represented. Although a grey scale image is commonly used in representing depth information of a certain

image, it is not as natural as in LazyBrush where colors represent nothing but colors. Consequently, the coloring process and region separation process can be combined effectively in LazyBrush but it might not be a good choice in our application.

We redesigned the process and as you can see in figure 3.6, the region separation process if independent of the depth adding process. In the very beginning of the process, the original hand drawn image and an image with color scribbles provided by the user are sent to LazyBrush. An image with different color regions are produced as the result.

It is only after we get the regions information that the user is allowed to manipulate the depth information.

To add depth to the original image is a very painful job. User have to find the boundaries of every region and paint every pixel in that region with a certain value. Fortunately, after the first step in our system, the hand-drawn image not only got colored but also segmented. After the image is segmented into different regions according to the color scribbles from the user input, we can add depth to each one of the regions easily. Because we now can assign a depth value to all the pixels in a certain region directly. There are two methods that we have proposed in the very beginning of the development process.

The first approach is simply by translating user movement into depth adjustment. For example, user clicks a region and drag upwards or downwards to indicate if they want to push the region inwards or to pull it outwards. The distance of the trace that the user dragged out indicates the distance by which user want the corresponding region to be pushed in or pulled out. This approach gives user more freedom which might not be a good thing. User tends to easily mess up something when they're given more freedom. However, user can change the depth of a region to a certain value easily.

The other approach is with the help topology sort, user specify the relationship between different regions. User drag an arrow from one region to another. The direction of the arrow indicates the relationship between the depth of different regions.

An example is given in the figure 3.5. The green arrow indicates that this is a legal operation. The red line indicates that there is a loop of relationships which is illegal. The red arrow is then dumped and user have to draw select a different operation.

After the depth information is gathered and a depth map is generated from the system. With the depth map and the original hand drawn image, we're able to extract normal information using the technique proposed in [4]. After the system generates the normal map, all of the preparation work is done.

It all comes down to the last process in which the original hand drawing is colored and shaded. This result image is displaced according to the depth information provided by the depth map so that we can get the left image and right image respectively.

Since we have two images to display, it depends on what type of stereoscopic display device is available.

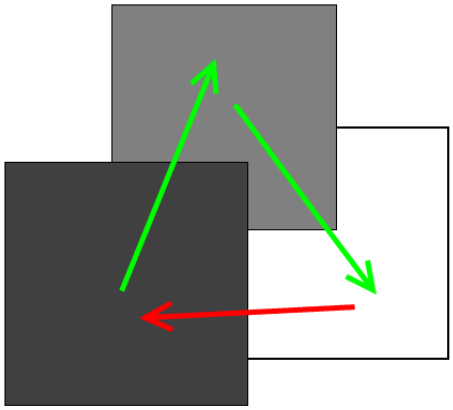


Figure 3.5: Example of using topology sort to detect the illegibility of the inter-relationship between regions

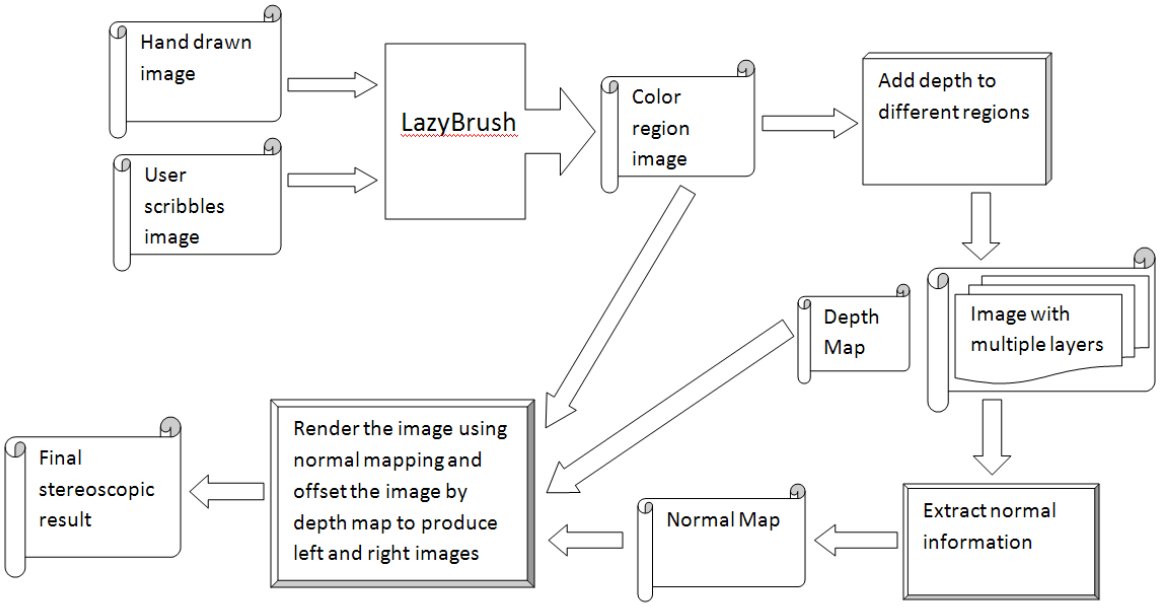


Figure 3.6: The modified design of the process

Chapter 4

Implementation

4.1 Image Segmentation

According to the artist who work in this field, they wanted a paint brush that has three features ideally,

- Optimal Boundary. The paint brush should fill as much area as possible by an optimal enclosing boundary (figure 4.1).

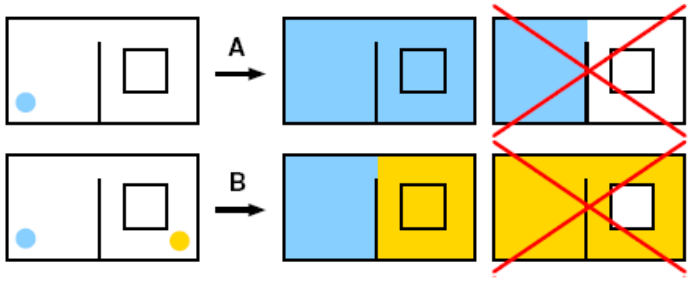


Figure 4.1: Optimal Boundary. Example from: [13]

- Soft Scribbles. The paint brush should resistant to imprecise placement (figure 4.2).

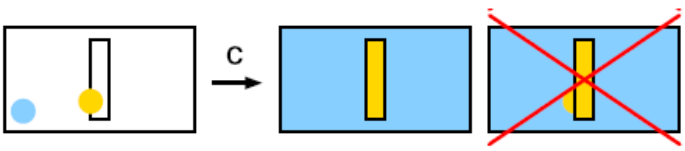


Figure 4.2: Soft Scribble. Example from: [13]

- Anti-alising. The optimal boundaries should be located in the place where the intensity is minimal (figure 4.3).

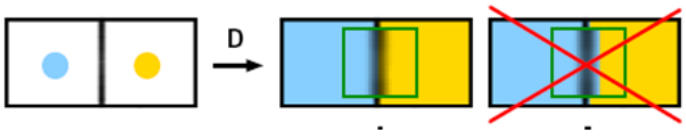


Figure 4.3: Anti-aliasing. Example from: [13]

To satisfy these requirements, Sykora et al. proposed a very unique approach in comparison to others. With the assumption that the pixels P are in a 4-connected neighborhood system N, a gray-scale image I is passed in as the original input. A set of user-provided non-overlapping strokes S with colors C are also passed in. To find how should the color labels be assigned to all pixels in the original image, all we need to do is to minimize the energy functions:

$$E\left(c\right) = \sum_{\{p,q\}?N} V_{p,q}(c_p, c_q) + \sum_{p?P} D_p(c_p)$$

The first term $V_{p,q}$ represents the energy of color discontinuity between two neighbor pixels p and q. The second term $D_p(c_p)$ indicates the energy of assigning a color label to a certain pixel.

The smoothness term is used to hide color discontinuities. Since the color discontinuities are often at pixels where the original image intensity is low (inside the dark outlines), the smoothness term is as follows,

$$V_{p,q}(C_p, C_q) \propto \begin{cases} I_p & for\ c_p \neq c_q \\ 0 & otherwise \end{cases}$$

As you can see, this term forces the color labels to push towards the dark outlines at which the value will be low even if the neighboring color labels are different. One issue need to be addressed is that when contrast between homogeneous areas and outlines are very low, problems can occur. This issue can be solved by using some nonlinear mapping to enhance the contrast. Alternatively, an Laplacian of Gaussian filter can be performed to detect the edges before the LazyBrush algorithm starts. An example is shown in figure 4.4.

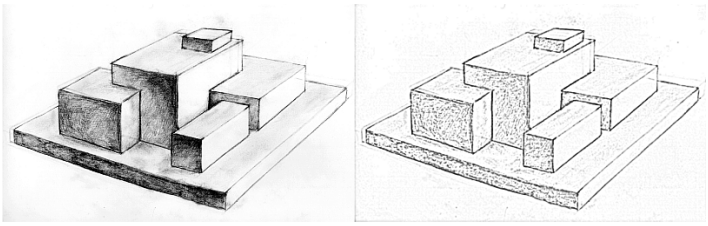


Figure 4.4: Image preprocessed by Laplacian of Gaussian filtering. Example from [13]

The data term, on the other hand, helps to determine how should the color labels be assigned if they are soft scribbles. The data term has a very high value K(around 4096) for pixels that are not covered by the original scribbles and a very low value (around 0) for the ones that are covered by the

original scribbles. For soft scribbles, the value is set to be a little bit lower than the highest value (λK, $\lambda = 0.95$).

In our application, the scribbles of the user are send to LazyBrush component and the image is segmented into regions. The color of each region is not something we focus in this project, the segmentation itself is what really matters here.

Unfortunately due to the time constrains, we didn't create an interactive interface for the LazyBrush component. User has to create an image with scribbles in advance and provide the system with the original hand drawn image with it to the system. After the segmentation process, each pixel in the image will be assigned with a label representing the index of the segment.

The segmentation works very well in the LazyBrush application however there are some aliasing effect in our application. If you look closely to the region marked by the red circles in figure 4.5, you'll notice these artifacts. The reason behind this is that, the aliasing artifact would always happen if we use the original energy function proposed in the paper [13]. However, this is trivial to the LazyBrush application due to the fact that all the aliasing artifacts are within the dark outlines where nothing will be noticed. However, in our application, this effect will appear when we generate the depth map from the segmentation image.

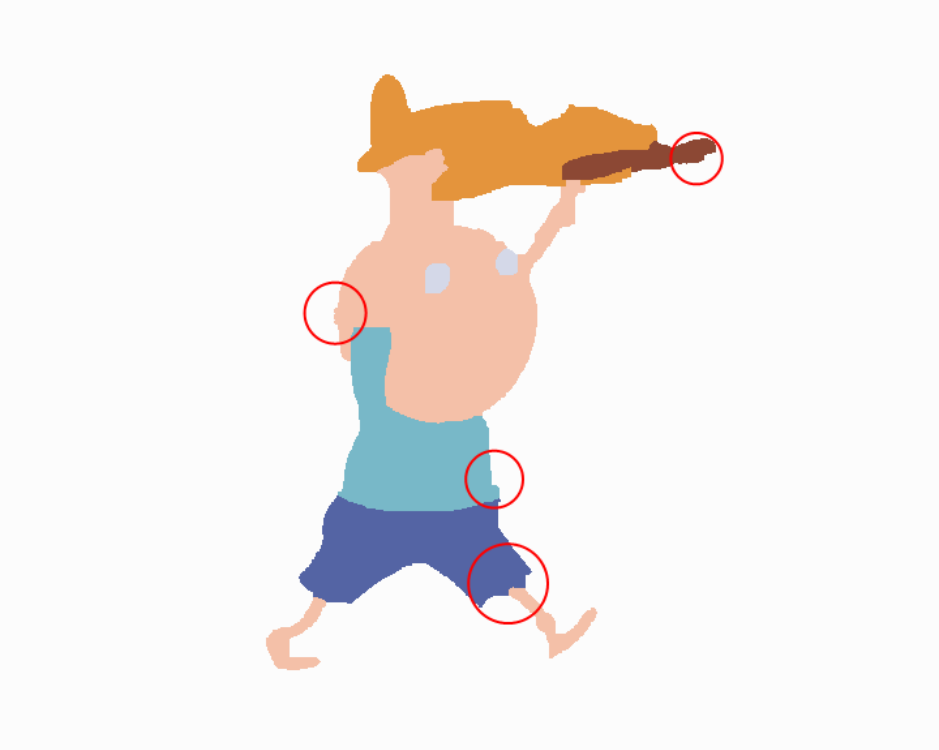


Figure 4.5: The aliasing artifacts when using the original energy function proposed in [13]

This artifact is very noticeable in our application but it can be removed by adjusting the energy function. However due to the time constrains we only used the original energy function in our ap-

plication and we removed these artifacts manually. In the future, an updated implementation of this component is definitely needed in order to speed up the process.

4.2 Depth Map Generation

As we mentioned in the previous chapter, there are two approach to assign depth to regions. The first approach allows user to assign depth to each region independently. This might not be a great approach since it is very hard for user to tell the depth just by looking at the gray scale depth map. We improved this approach by telling user the exact depth value of each region when they move the mouse cursor onto it. Unfortunately this is still doesn't work very well.

The second approach helps user to specify spatial relationships between regions. In this way, unexperienced user can work with system easily. When user tries to assign depth illegally to a certain region, the system would complain and guide the user to the correct solution. However, the detailed depth value are automatically assigned by the system and user wouldn't be able to control it. It would be much better if the two approaches can be combined such that user can be guided when assigning depth values to different regions yet still control the exact depth values.

Due to the time constrains, we only implemented the first approach however we modified it a little so that the second approach can be implemented and added in the future. Our original implementation is using mouse drag to determine what the depth value should be for a certain area. However, this is in conflict with the second approach which also uses mouse drag operations to determine depth. Therefore we adjusted the implementation such that, instead of using mouse drag, we used mouse wheel rotation to control the depth.

The approach that the depth information is assigned relatively can be then added into the system with no extra requirement. In fact, we left the stub in our program for future implementation.

4.3 Image Enhancement to show spatial differences

Image enhancement, as we discussed earlier, can provide representations of spatial differences. The technique behind this is very easy to implement and the overhead of this technique is very small. Consequently it can be added either to the editing process or the final result to improve the effects.

After the depth information of the objects are added by the user, we have to find a better way to represent the spatial relationship between neighboring regions. Using the technique proposed in the paper [8], we can achieve this.

This component is implemented as a rendering pass using HLSL shader language and it runs on the GPU. It is composed of two separate processes. First, the grey-scale image that represents the depth information (depth map) is passed in as input image. We then calculate a Gaussian blurred version of this depth map G*D.

After that, we use the original depth map to subtract the Gaussian blurred version of the same image. The result we get from this operation is the spatial importance function. This spatial impor-

(a) Original depth map (b) Low-pass filtered depth map

Figure 4.6: Pre-processes for the spatial importance calculation

tance function indicates the foreground-background relationship between every region. If the value of the spatial importance function is greater than 0, the region is in the background comparing to its neighboring region and vice versa.

$$\Delta D = G * D - D$$

We can leave the values that are less than or equal to zero in the spatial importance function and get rid of the ones that are greater than zero. The result we get will be the contrast enhancement map which we can use along with the original image to produce the final result.

There are two shaders involved in this process

- PseudoGaussian.fx is our low-pass filter that blurs the image. Originally, this is done by the GaussianBlur. However, in practice, we found that the resulting effect using Gaussian as low-pass filter is not as good as this "fake" Gaussian filter.
- ContrastEnhancement.fx takes in three textures as input, the original image, the depth image and the low-pass filtered depth image. It calculates the spatial importance function ΔD and process it further into the contrast enhancement map by getting rid of the value that's greater than zero.

As we can see in the figure 4.6, the contrast enhancement map is calculated from the first two images 4.6a, 4.6b by subtracting them from each other and removing the values that are greater than

zero. Then we apply the contrast enhancement map 4.7a. The final result is simply synthesized by adding the original hand drawn image with the contrast enhancement map.

(a) Contrast enhancement map (b) Final result

Figure 4.7: Contrast enhancement map and the contrast enhanced hand-drawn image

4.4 Extract Normal Map

After the depth map is generated, the next would be the generation of the normal map. As we discussed in the second chapter, the technique introduced in the paper can approximate lighting on 2D drawings. The key idea behind this technique is that the z-component of the normals along the edge of a curved surface must be zero. Imagine a scene in Orthographic projection system. All the vertices on the boundaries of an object must have its normal in a 2D space (only has x component and y component, z component is always zero).

Although in perspective projection, the key idea we just mentioned above doesn't apply, scaling can be added to achieve the same result in perspective projection.

Since we know that there are only two component for the normals that belong to the points on the boundaries, we can calculate those normals in 2D space simply by calculating their gradients.

To achieve better performance, we implemented this component using HLSL which means that our gradient calculation is performed on the GPU. The calculation of gradient of a certain point on the edge is very easy. We only need to combine the two orthogonal partial derivatives (vertical and horizontal) to calculate the orientation like this,

$$\varphi(i,j) = atan2(\frac{\delta f(i,j)}{\delta j}, \frac{\delta f(i,j)}{\delta i})$$

However, if we directly perform the gradient calculation on the original hand drawn image, we will get something like this (figure 4.8). This is obviously not what we want as you can see every

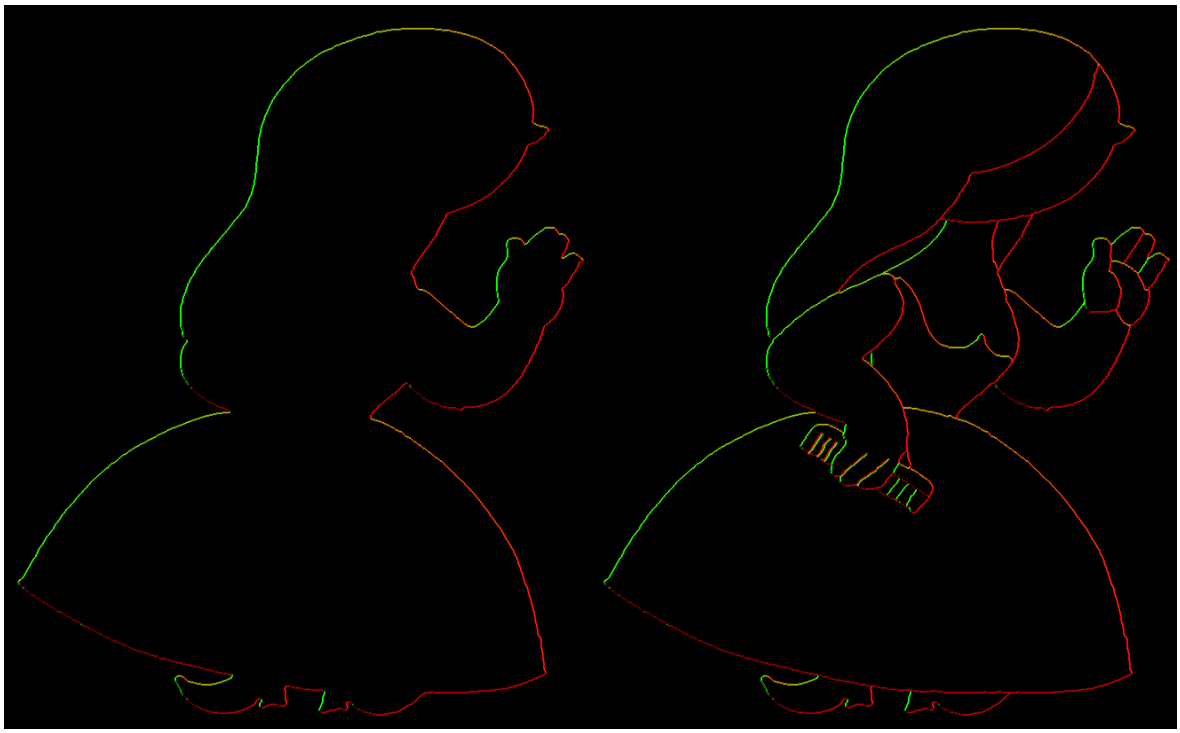
Figure 4.8: The result after performing the gradient calculation directly on the original image

boundary produces two colored contours. What we can do is instead of calculating the gradient of the boundary, we can fill in the regions with color that's same as the boundary and then perform the calculation. In this way, there will be only one gradient change per boundary.

After we apply this gradient calculation to the image that we modified, we'll get an image like this (figure 4.9a). Apart from this, we also need to calculate the normals for each region independently. This can be done easily by performing the gradient calculation directly on the depth map, which is generate from the previous steps.

After we calculated the gradient of the boundaries, we can construct our original normal map by setting all of the components on these boundaries to be zero. We then need to interpolate these values across every region. The [4] paper suggested us to use a damped spring model to interpolate the normal values.

To be more specific, as the paper suggested, the pixels on the boundaries (which are already known) should be regarded as fixed points and the rest of the pixels are relaxed across the field. The



(a) The original image (b) Anaglyph of the original image

Figure 4.9: The gradient map calculated from the modified original image and the depth map

difference between the color values of neighboring pixels is equivalent to the force that between points on the damped spring system. The bigger the difference between a pixel and its neighboring pixels, the bigger the "force" it receives to change color towards its neighbors' colors.

One of the great things about this model is that the computation can be performed iteratively instead of try to get it right in one go. The color values of each pixel in this field changes in each iteration towards a value that's more correct. In every iteration, the following calculation is performed.

$$V^new_{i,j} = d \cdot V_{i,j} + k \cdot (P_{i-1,j} + P_{i+1,j} + P_{i,j-1} + P_{i,j+1} - 4P_{i,j})$$

$$P^new_{i,j} = P_{i,j} + V^new_{i,j}$$

$V_{i,j}$ stands for velocity field in which stores the value of the "forces" to change the colors of a certain pixel. During every iteration, the velocity is calculated by subtracting the color values of the current pixel from the value of its neighboring pixels and scaled by a value k (k=0.4375 suggested in the paper). And the new velocity value computed by adding up the previous value scaled by d (d=0.97 suggested in the paper) and the value we just calculated.

The iterations will keep running until the mean squared velocity per pixel reaches an acceptable

error tolerance which means a balanced state where all the points are moving in a very small range.

However, this approach was originally designed to simulate a dynamic environment and when it is translated into this situation the results we got are not very convincing. This could have been an implementation issue.

There is a better and standard approach to do this. Consider that in a perfectly interpolated image, the rate of change of the gradient of each pixel should be zero. And hence if we calculate the Laplacian (2D isotropic measure of the second spatial derivative) of the image, we should get all zeros.

This forms up a linear system and if we solve this linear system, we'll get a perfectly interpolated normal map. A linear solver can be easily find on the internet. However, due to time constrains and my lack of knowledge in this area, our project only implemented the interpolation technique proposed in the [4] paper.

4.5 Displacement of Regions

Originally, this step of the whole process was considered to be the easiest one. Since we already have the depth map, which has the depth value of every single pixel in the image, we should be able to scale all of the depth value by a certain value to create a displacement map. Unfortunately, things are not that easy. If we use the displacement map generated by simply scaling the depth value, we will get some kind of result like this (figure 4.10).

You might have noticed that the image looks like it's corrupted. What happened was, the regions on the original image gets displaced by different values, some of these regions will cover other regions so that they gets overlapped with each other. And at some other parts of the image holes will appear. Because we only have one image from a single view, artifacts like holes between regions will always appear when we try to generate an image from a different view unless you use approach like interpolation between regions. But we should be able to minimize these artifact.

If we pre-calculate the displacement map such that the regions with smaller depth value (closer to the viewer) will not be covered by a further region if they need to overlap with each other, we can minimize that effect so that less artifacts will be noticed. For every pixel, we scale the depth value on this pixel to produce the displacement value. In addition to that, we keeps record of the depth value of this pixel so that when some other pixel gets displaced to this point, a comparison between the depth will be performed.

In this way, we can get a new displacement map and hence a better new result (figure 4.11).

4.6 Stereoscopic 3D Rendering

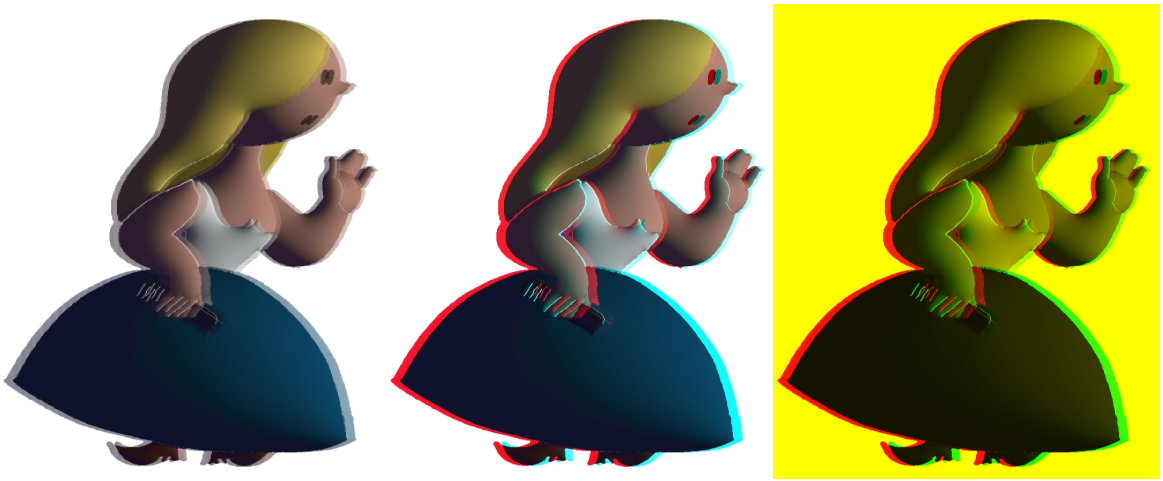
After we displace the image, we can get two images for both the left eye and the right eye. These two images can be the input of any stereoscopic 3D rendering techniques and generate the final effect by combining them together. In this application, we used anaglyph as our main display devices for its simplicity.

Figure 4.10: Problem using the scale depth value as displacement value

The left and right images are passed into the shader and the shader uses two color filters to filter them into two different colors. The default one in our application is red and green which produces results that can be viewed by red and green glasses.

Just to prove that our application can be easily extended to support other type of stereoscopic 3D effect rendering, here are some examples. The image on the left is rendered for 3D LCDs, the one in the middle and the one on the right are both anaglyphs but filtered with different color masks.

Figure 4.11: Artifacts reduced results of displacement



(a) 3D LCD (b) Red Cyan Anaglyph (c) Red Green Anaglyph

Figure 4.12: Different stereoscopic rendering technique applied to the same two input images

Chapter 5

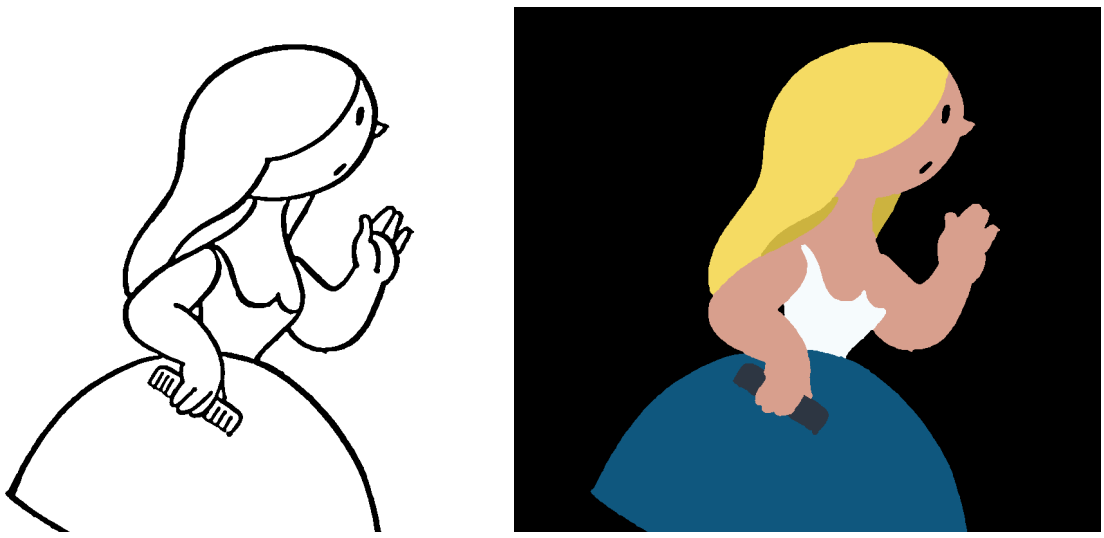
Result and Evaluation

5.0.1 Results

Note: All the anaglyph images are rendered for red-green glasses.

Here are some example of using our system to generate realistic stereoscopic 3D effect from a single hand drawn image. Some of the images generated by the process in the middle are here as well.

For the first example, the original hand drawn image (figure 5.1a) is colored using the LazyBrush technique (figure 5.1) to generate the image with color regions (figure 5.1b).

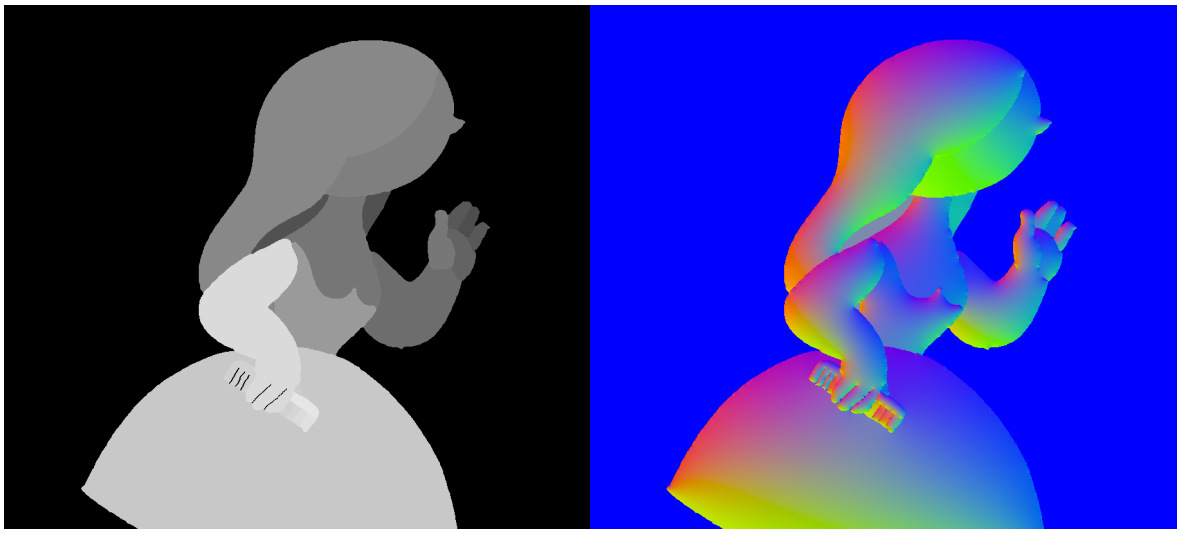


(a) Original image

(b) Colored image

Figure 5.1: The preparation stage, the original image is analyzed and colored use the LazyBrush technique.

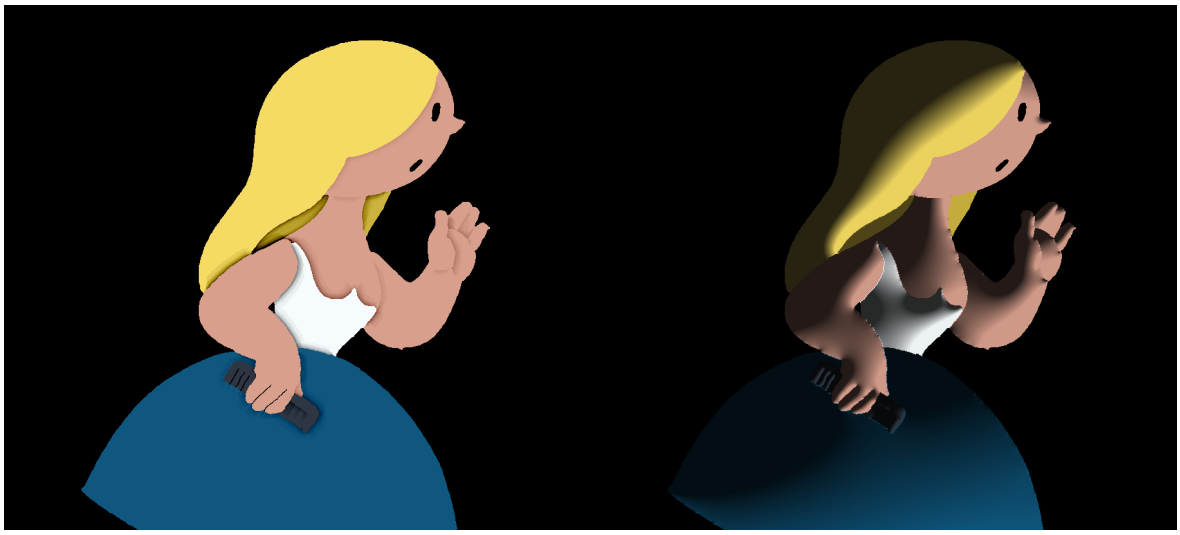
The depth of every region is added by the user to create the depth map (figure 5.2a). The normal map is then added using the Lumo technique ([4]) (figure 5.2b).



(a) Depth map, image from unpublished paper [12] (b) Normal map, image from unpublished paper [12]

Figure 5.2: Depth map is subsequently generated with a few mouse operations by the user. The normal map is generated as well

The result of image enhancement proposed in [8] (figure 5.3a) and the image (figure 5.3b) rendered using normal mapping.

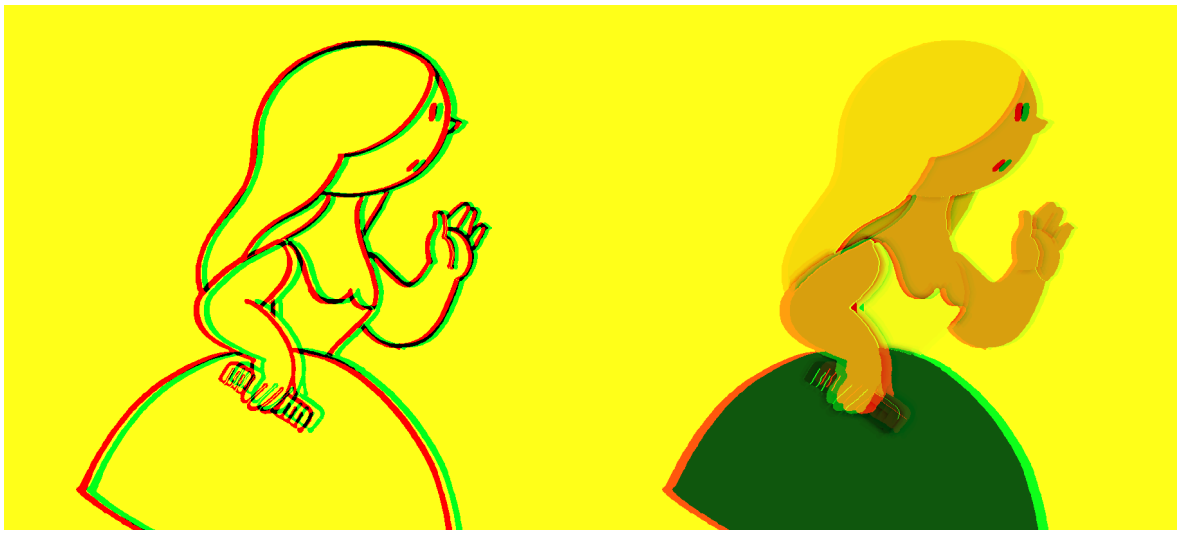


(a) Enhanced image (b) Normal mapped image

Figure 5.3: The enhanced image using the unsharp masking technique and the image rendering using normal mapping

After depth map is generated, we calculate the displacement map and displace the image according

to it. We tried several different images as comparisons. The one on the left (figure 5.4a) is converted to anaglyph from the original hand drawn image. The one on the right (figure 5.4b) is converted from the enhanced image.



(a) Anaglyph from Boundaries (b) Anaglyph from enhanced image

Figure 5.4: The rendered anaglyph, the left image is rendered from the original hand drawing, the one on the right is generated from the contrast enhanced version of the image

The final result, our goal of the system is shown in figure 5.5. It is converted from a normal mapped image.

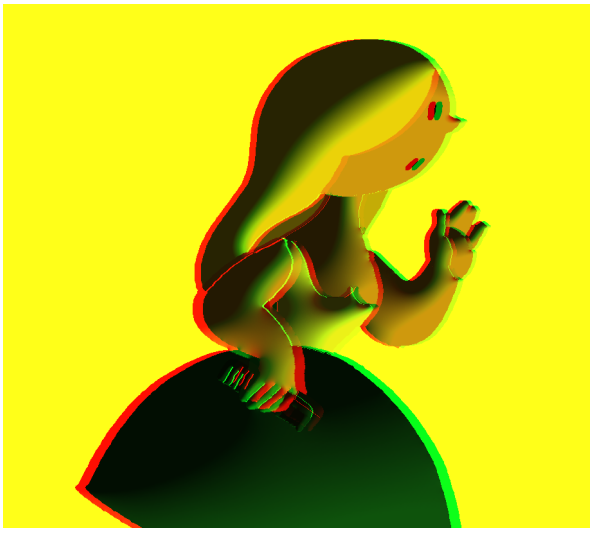
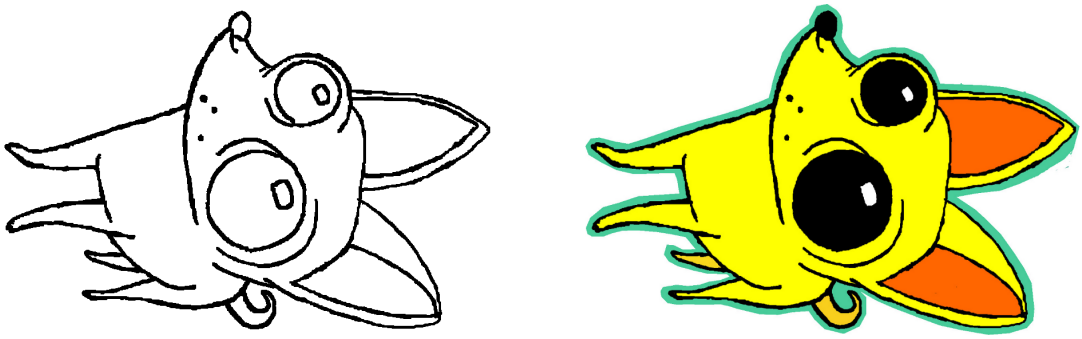


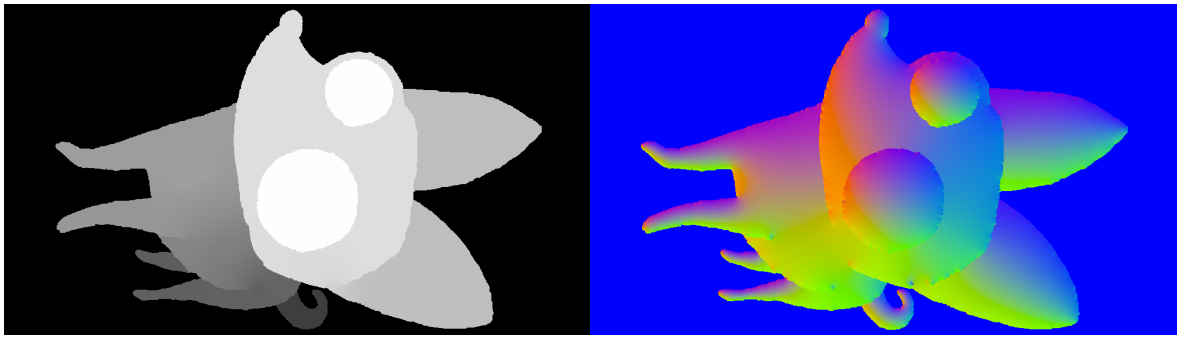
Figure 5.5: The final result of rendered anaglyph with normal mapping

More examples are illustrated in the following pages.



(a) Original image, image from unpublished paper [12] (b) Colored image, image from unpublished paper [12]

Figure 5.6: The preparation stage, the original image is analyzed and colored use the LazyBrush technique.



(a) Depth map, image from unpublished paper [12] (b) Normal map, image from unpublished paper [12]

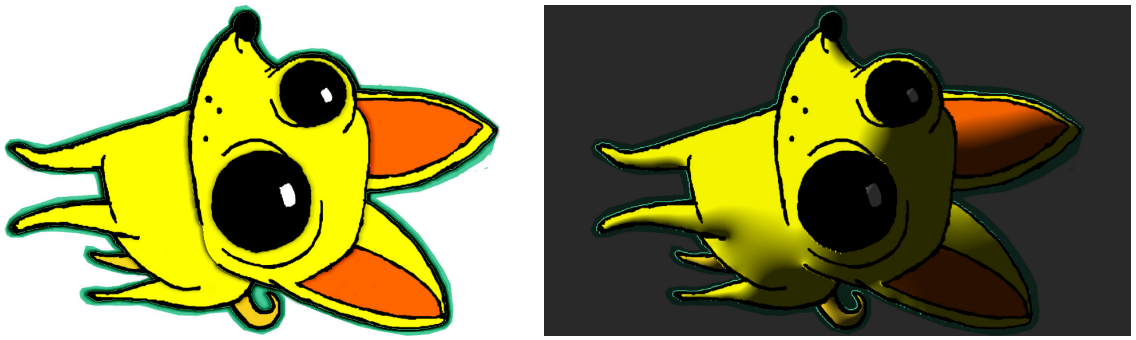
Figure 5.7: Depth map is subsequently generated with a few mouse operations by the user. The normal map is generated as well

Here are another two examples, we only show the final result corresponding to the hand drawn image.

5.0.2 Evaluation

The results we generate from the sample hand drawn images are quite convincing. The anaglyph generated directly from the original hand drawn image gives a very week depth perception. After we enhance the image using the unsharp masking technique proposed in [8], the quality of the anaglyph is largely improved. When we render the image using bump mapping with automatically generated normal map, the anaglyph instantly pops up and the feel of smooth depth transition appear.

We did a small perception experiment with all the sample results and we found some interesting things. All of the four examples we used above received similar feedback. The averaged statistics are



(a) Enhanced image (b) Normal mapped image

Figure 5.8: The enhanced image using the unsharp masking technique and the image rendering using normal mapping

(a) Anaglyph from Boundaries (b) Anaglyph from enhanced image

Figure 5.9: The rendered anaglyph, the left image is rendered from the original hand drawing, the one on the right is generated from the contrast enhanced version of the image

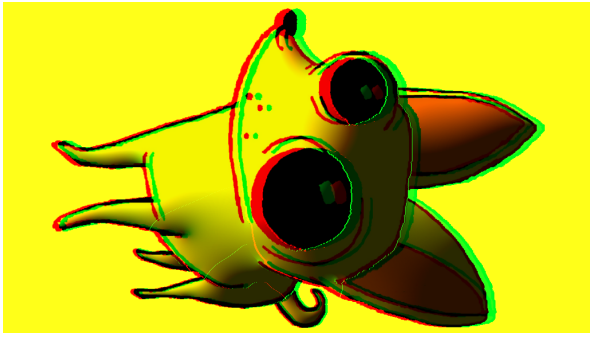


Figure 5.10: The final result of rendered anaglyph with normal mapping

displayed in the following table.

Image 1 is the original hand drawn image. Image 2, 3 and 4 are the anaglyph images generated from the original image, the enhanced image and the normal mapped image, respectively. For the

(a) The original image (b) Anaglyph of the original image

Figure 5.11: The original hand drawn image and the anaglyph converted from it

Question	image1	image2	image3	image4
how noticeable is the depth feel (0-10)	0	6	9	8
how strong is the blobby feel (0-10)	0	0	0	8
what type of structure does it use	1	1	1	2(2/10) and 3(8/10)

Table 5.1: The results of the perception experiment

third question, the three types of structures are:

1. Layers of flatness
2. Smooth depth transition
3. Combined

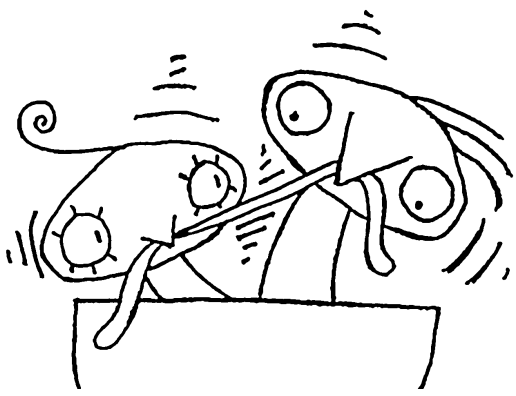
As we can see in the result, the anaglyph generated from the original image gives a very week depth feel. With image enhanced by unsharp masking, the depth feel is dramatically increased. The optimization performed on the 2D images does have a very big influence on stereoscopic 3D results.

Another interesting point is that 80 percent of the people can feel the blobby effect in image4 but few of them thinks that it's organized in a structure where all the depth information are smoothly

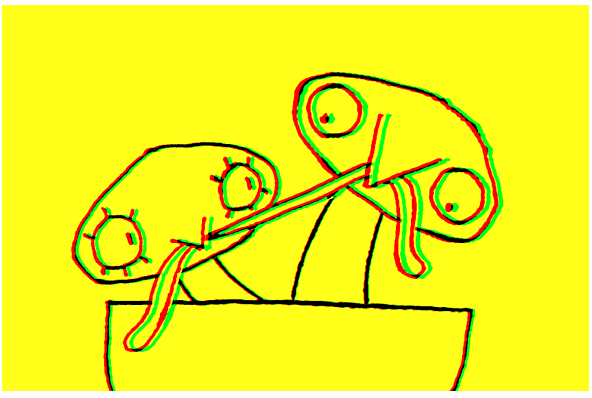
(a) The stereoscopic image converted from the enhanced image

(b) The stereoscopic image converted from the normal mapped image

Figure 5.12: The anaglyphs converted from the enhanced image and the normal mapped image



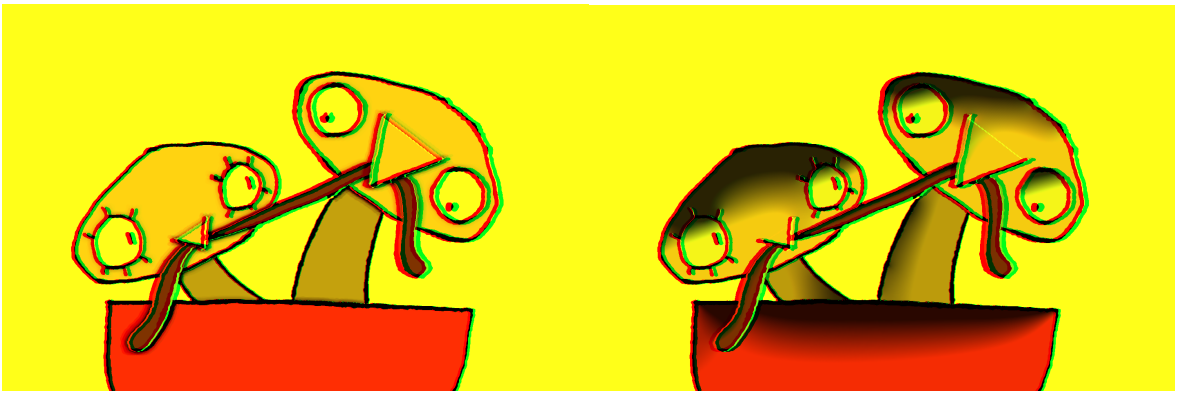
(a) The original hand drawn image, image from unpublished paper [12]



(b) The stereoscopic image converted from the original hand drawn image

Figure 5.13: The original hand drawn image and the anaglyph converted from it

transited. Most of the people noticed the sharp depth transition between different layers and they also noticed the smooth transition on a single region. This might have been an implementation issue.



(a) The stereoscopic image converted from the enhanced image

(b) The stereoscopic image converted from the normal mapped image

Figure 5.14: The anaglyphs converted from the enhanced image and the normal mapped image, respectively

Using a differently interpolated normal map might will solve this issue.

Chapter 6

Future Work

LazyBrush can help us separating the image into different regions by color scribbles and the result of using it to color an image is very impressive. However, in our application, the segments that LazyBrush generated are used for depth modification as well and some of the artifacts reveals. There are some aliasing artifacts on the boundary of the segments output from LazyBrush. Originally, none of these aliasing effect would appear since they all hide in the dark boundaries. The aliased outlines will appear in our application due to the reason that they are depth information rather than colors now hence couldn't hide in the dark outlines.

To achieve better visual effect, a modification need to be done to the original LazyBrush such that it includes the boundaries when used to separate the image. Once this modification is done, the aliasing effect will be eliminated.

For depth adding process, instead of using independent user operations, the other approach using topology sort should be implemented in addition to the one we already have. Using this technique, the depth information can be added more interactively and much easier for users. Combing both approach can give the artist full control over the depth map and the process to modify the depth information should still be very easy.

In this application, we used normal mapping technique in the rendering stage. This approach enables us to calculate the lighting in real-time such that the artist can change the lighting interactively. As we discussed earlier, sphere map enables artist to render the scene with a pre-rendered unit sphere. The artistic styles are very easily controlled in this way although the speed would probably decrease. We can implement Sphere Map Illumination technique so that artist can have easier and better control of the artistic properties of the rendered scene.

One common problem about anaglyph is ghosting. It is caused by ill-filtered images. In other words, if the color filters we used for creating the anaglyph don't match the color of the classes, ghosting will appear. But with the improving popularity of other displaying devices, we can implement other stereoscopic 3D rendering techniques to switch to other display devices such as screen-synchronized shutter glasses or 3D LCD. We have shown the potential of using our system to provide enough information to generate stereoscopic 3D images for other devices. However more work need to be

done.

In the rendering phase of our application, we always use normal mapping to illuminate the drawing first and then displace the image is not the perfect solution. Although it produces a convincing results, the approach isn't quite right. The features(highlight and the boundaries) on a region should have different depth and consequently be displaced to different positions. In perfect situation, the images should be rendered twice using normal mapping with the correct depth information. Our approach, for simplicity only did it once. Improvement can be made with regard to this issue.

Alternatively, a polygonal mesh can be employed to replace the 2D textures such that we don't need handle the displacement map and the problem above will be solved naturally. The stereoscopic 3D rendering will just involve moving the camera position and render the scene twice.

Chapter 7

Conclusion

With the improvement of technologies in stereoscopic 3D field, more and more people started embracing games or movies in stereoscopic 3D. And recently there has been a great trend among animation companies to go back to classic 2D hand drawn animation. Companies like Walt Disney animation studio has been trying converting their hand drawn animations into stereoscopic 3D to provide the audience with a feel of the classic content.

However, there are little research in converting hand drawn animation into stereoscopic 3D. And other techniques like Teddy or SmoothSketch don't give users much control over the final result although works pretty well. And most of all, nearly all of these techniques involves a great deal of analysis and geometry transformations, etc.

Our approach, which borrows the same idea of bump mapping that detailed view can be created from simple objects. We developed a easy-to-use tool based on LazyBrush that allows user to separate image into separate regions and assign depth information onto each one of them. Although the depth information is only able to create 3D structures such that they look like layers of flatness, the final effect can be quite impressive once we render each region with normal mapping or other similar technique to produce detailed rich content.

The final result is quite convincing however there are still improvements to be made. The rendering of each region is only done once in our approach and this can cause the regions to look flat than it should look like if the image is rendered twice properly with different displacement value.

Appendix

.1 TGA File Format

.2 Class Diagram

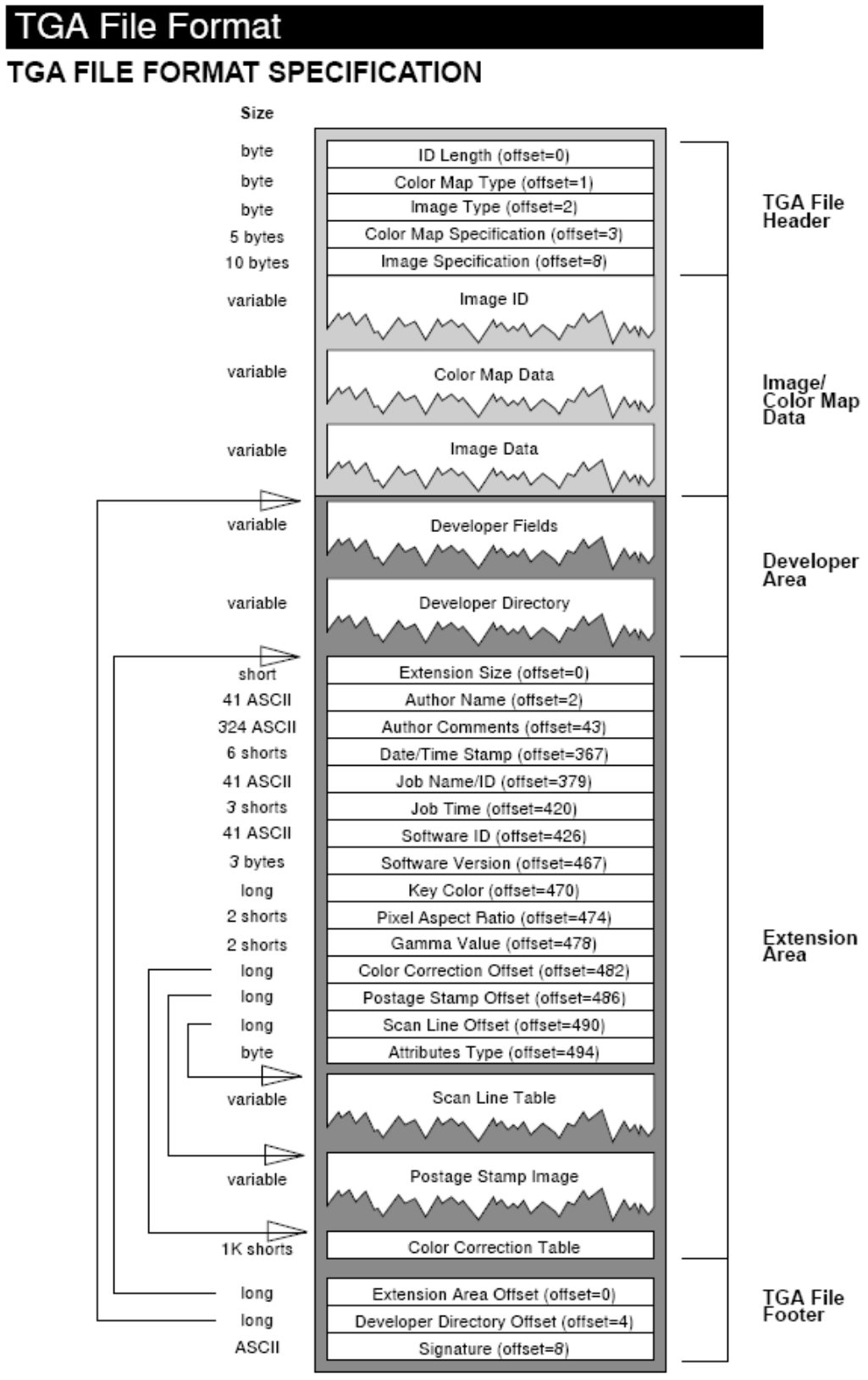


Figure 1: TGA File Format

Figure 2: Class Diagram

Bibliography

[1] Li Zhang Guillaume, Li Zhang, Guillaume Dugas-phocion, Jean sebastien Samson, and Steven M. Seitz. Single view modeling of free-form scenes. In *In Proc. of CVPR*, pages 990–997, 2002.

[2] Youichi Horry, Ken-Ichi Anjyo, and Kiyoshi Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 225–232, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[3] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[4] Scott F. Johnston. Lumo: illumination for cel animation. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 45–ff, New York, NY, USA, 2002. ACM.

[5] Sing Bing Kang. Depth painting for image-based rendering applications, 1998.

[6] Sing Bing Kang and Huong Quynh Dinh. Multi-layered image-based rendering. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 98–106, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[7] Olga A. Karpenko and John F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, 2006.

[8] Thomas Luft, Carsten Colditz, and Oliver Deussen. Image enhancement by unsharp masking the depth buffer. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1206–1213, New York, NY, USA, 2006. ACM.

[9] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John F. Hughes. Skin: A constructive approach to modeling free-form shapes. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 393–400, August 1999.

[10] Lakshman Prasad. Morphological analysis of shapes. *CNLS Newsletter*, 139:1–18, 1997.

[11] Yingge Qu, Tien-Tsin Wong, and Pheng-Ann Heng. Manga colorization. *ACM Trans. Graph.*, 25(3):1214–1220, 2006.

[12] D. Sýkora, D. Sedláček, S. Jinchao, J. Dingliana, and S. Collins. Cartoon pop-up using sparse depth inequalities. Unpublished.

[13] Daniel Sykora, John Dingliana, and Steven Collins. Lazybrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum*, 28(2):599–608, 2009.

[14] Gang Zeng, Yasuyuki Matsushita, Long Quan, and Heung-Yeung Shum. Interactive shape from shading. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 343–350, Washington, DC, USA, 2005. IEEE Computer Society.