

3D Imaging of Spherical Objects

by

Chris Tsang

Gurinder Kler

Final report submitted in partial satisfaction of the requirement for the degree of

Bachelor of Science

In

Electrical and Computer Engineering

in the

Faculty of Engineering

of the

University of Manitoba

Faculty Supervisor:

Dr. Richard Gordon, Professor, Department of Radiology, University of Manitoba
Adjunct Professor of Electrical & Computer Engineering

Spring 2002

© Copyright 2002 Chris Tsang, Gurinder Kler

Abstract

The purpose of this design project is to develop platform independent software to manipulate 2D images and to create a 3D image of a spherical object from six different 2D views. Each of the six input images corresponds to one side of a cube. Software was developed in the ImageJ environment which allows for development of Java encoded plugins. The design project resulted in the creation of a GUI (Graphical User Interface) that allowed for manipulation of 2D images, including the displaying of six consecutive images over time. A 3D spherical object was created from six 2D images. The functionality of the 3D software can be improved by optimizing the calculation of greyscale algorithm, calibrating images to become perpendicular, reducing computing time, rotating the spherical object, and using third party software to create a realistic 3D spherical object.

Table of Contents

Abstract	ii
Table of Contents	iii
List of Tables	v
List of Figures	v
Contributions	vi
Acknowledgments	vii
Chapter 1: Introduction	1
Chapter 2: ImageJ	2
2.1 Where ImageJ is being used.....	3
2.2 Why use ImageJ?	3
2.2.1 Evaluation of OpenGL.....	3
2.2.2 Evaluation of Java3D.....	4
2.2.3 Evaluation of ImageJ	5
2.3 Chapter Summary	5
Chapter 3: 2D Manipulation of Images.....	6
3.1 2D Graphical User Interface	6
3.1.1 Functions.....	6
3.2 Combine Stacks Plugin	7
3.2.1 Combine Stacks Algorithm.....	7
3.2.2 Proper Use and Limitations.....	10
3.3 Chapter Summary	10
Chapter 4: 3D Creation of Spherical Objects.....	11
4.1 3D Software Assumptions	11
4.2 Step 1: Load Images.....	11
4.3 Step 2: Center Images	11
4.3.1 Limitations	12
4.3.2 An Alternative Centering Approach	12
4.4 Step 3: Find Radius of Each Image.....	12
4.5 Step 4: Calculate Third Coordinate.....	13
4.6 Step 5: Calculate Weighted Greyscale.....	15
4.7 Step 6: Display each of the Six Images with new Greyscale.....	19
4.8 Step 7 Rotate Image Object	20

4.9	Chapter Summary	20
Chapter 5: Future Work		21
5.1	Improving Calculation of Greyscale	21
5.2	Calibrating Images to be Perpendicular	22
5.3	Reducing Computing Time	22
5.4	3D Rotation	24
5.4.1	3D Representations	24
5.4.2	Rotation about the Coordinate Axis	25
5.4.3	Rotation about an Arbitrary Point	26
5.4.4	Section Summary	26
5.5	Use of Commercial Software	26
5.5.1	Section Summary	28
5.6	Chapter Summary	28
Chapter 6: Application to 3D Spherical Objects		29
6.1	Procedure	29
6.2	Chapter Summary	30
Chapter 7: Conclusion		31
Appendix A: 2D Imaging Java Code		viii
Appendix B: 3D Creation of Spherical Object - Java Code		ix
Appendix C: Computation Time Tables		x
References		xii
VITA I		xiii
VITA II		xiv

List of Tables

Table 2-1. Imaging Environments	3
Table 4-1. Third Coordinate Value.....	14
Table 5-1. Processing Time.	23
Table 5-2 Processing Time per View.....	23
Table C-1. Processing Time per View on AMD Duron 850MHz	x
Table C-2. Processing Time per View on PIII 850MHz.....	x
Table C-3. Processing Time per View on AMD K6 300MHz.....	x

List of Figures

Figure 2-1. ImageJ Application Screen.....	2
Figure 3-1. Two Dimensional GUI.....	6
Figure 3-2. Flow chart of Combine Stacks plugin	8
Figure 3-3. Placement of pictures.	9
Figure 3-4. Combined image stacks of a basketball.	9
Figure 4-1. Coordinate Structure of Images.....	15
Figure 4-2. Original Views.	17
Figure 4-3. Views with Weighted Greyscale.....	18
Figure 4-4. Views Reconstructed from Weighted Greyscale views.	19
Figure 5-1. Blocked Pixel on Sphere	22
Figure 6-1. Front view of the proposed system.	29

Contributions

The main accomplishment of this design project was the creation of Java plugins for ImageJ that allow users to manipulate 2D images and create a 3D image of a spherical object from six different 2D views. This software will aid in the research of salamander embryonic development.

Chris Tsang was responsible for the creation of the two dimensional GUI and the code for the StackCombiner.java plugin which allows six image stacks to be combined into a single image window. He was also involved in implementing the code for centering the images and for finding the radius of the spherical object. For the final report, he was responsible for the sections that pertained to the work mentioned before. He also wrote Chapters 5 and 6 of the final report with collaboration from Gurinder Kler.

Gurinder Kler was responsible for implementing the code to calculate the third coordinate and new greyscale value for each of the six images. He was also responsible for displaying the newly acquired greyscale values in a new image window. For the final report, he wrote about the sections that pertained to the work described above and was involved in writing Chapter 2 of the final report. Gurinder also collaborated with Chris Tsang on Chapters 5 and 6 of this report.

The remaining sections of the report (introduction, abstract, contributions) were written by both team members.

Acknowledgments

The authors of this report would like to thank Dr. Richard Gordon for his guidance throughout the design of this project and for giving them the opportunity to develop imaging skills and learn the Java programming language. They would also thank Terry Walker and Steve Shaw for their insight on commercial 3D software.

Many people have heard the authors' discussions on 3D imaging in personal talks at school, home and at the bar. The authors would like to thank these people for listening to their difficulties and for any suggestions they may have given us, no matter how useful or useless they were. In no particular order these people include Jesse Brar, Cam Melvin, Peanut, Leslie Yeow, Rodeo, Leah McCartney, Gilbert Brunette, Marvin Ballesteros, Elias Saghbini, Rambo, the Tsang Family, the Kler Family, Webster (Bartender at the Beach), Ian Ponce, Jason Masesar, Ben Yue, Jane Quioque, Seagram's spirits, Smirnoff spirits, Canada's Gold Medal winning 2002 Men's Olympic Hockey Team, the sport of basketball, and Spalding.

Chapter 1: Introduction

This report will give a complete discussion of the development involved in the 3D imaging project. The project required that six two dimensional images, representing the six different views of a sphere, be manipulated so that the images can be properly used to create a three dimensional representation of the sphere. The motivation behind creating a 3D spherical object from 2D images is to determine if a cost-effective platform independent application can be created so that this can be used for research into salamander embryonic development.

The report is organized in the following chapters:

- **Chapter 1: Introduction** provided an overview of the report and a summary of the chapters in this report
- **Chapter 2: Image J** describes the Image J application used in the project and an evaluation of other applications.
- **Chapter 3: 2D Manipulation** discusses the work that was done on the design of a GUI and the plugin that was created to display six image stacks in one image window
- **Chapter 4: 3D Creation of Spherical Objects** explores the development of the three dimensional object and the required manipulations involved before a 3D object can be created
- **Chapter 5: Future Work** gives an in depth description of improvements to the 3D imaging software.
- **Chapter 6: Application of 3D Spherical Objects** discusses the application of this project within the 4D Imaging Project.
- **Chapter 7: Conclusion** provides a summary of what has been achieved in this project.

Chapter 2: ImageJ

ImageJ is a platform independent image processing application written in the Java programming language. It has been developed and maintained by Wayne Rasband of the NIH (National Institutes of Health) Research Services Branch. ImageJ can be run as an online applet or as a local application, on any computer with a Java 1.1 or later virtual machine. ImageJ has many image manipulation features and it allows for plugins to be written in Java to expand on the features available.

A screen shot of ImageJ can be seen in Figure 2-1 below.

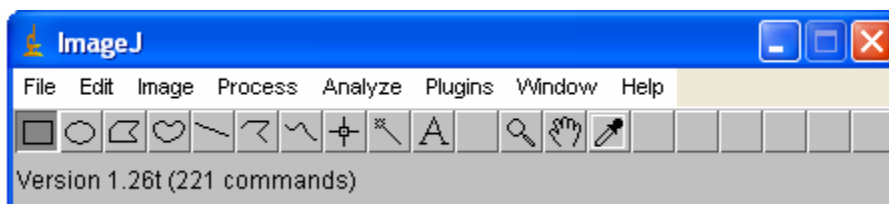


Figure 2-1. ImageJ Application Screen¹.

ImageJ has a long list of features. Some of its image processing capabilities are listed below¹:

- It can display, edit, analyze, process, save and print 8-bit, 16-bit and 32-bit images.
- It can read many image formats including TIFF, GIF, JPEG, BMP, DICOM, FITS and "raw".
- It supports "stacks", a series of images that share a single window.
- It is multithreaded, so time-consuming operations such as image file reading can be performed in parallel with other operations.
- It can calculate area and pixel value statistics of user-defined selections.
- It can measure distances and angles.
- It can create density histograms and line profile plots.
- It supports standard image processing functions such as contrast manipulation, sharpening, smoothing, edge detection and median filtering.
- It does geometric transformations such as scaling, rotation and flips.
- Image can be zoomed up to 32:1 and down to 1:32. All analysis and processing functions are available at any magnification factor.
- The program supports any number of windows (images) simultaneously, limited only by available memory.

¹ Source ImageJ website, <http://rsb.info.nih.gov/ij/>

- Spatial calibration is available to provide real world dimensional measurements in units such as millimeters.
- Density or gray scale calibration is also available.

To expand on the current features new plugins can be written in Java. ImageJ is an open source API (Application Programming Interface). It was designed to allow one to use its current features in expansion plugins.

2.1 Where ImageJ is being used.

Many people dealing with captured images are using ImageJ. Since it is being developed by the National Institutes of Health, a large majority of the users are from the medical imaging field. A search of ImageJ on the web will lead one to many sites where ImageJ is being used.

2.2 Why use ImageJ?

There are many imaging environments available, many of them being open source and available for free. Not all environments were evaluated, as this could be a project in itself. Options that were considered are OpenGL, Java3D and ImageJ. Some of the notable environments are listed in Table 2-1.

Table 2-1. Imaging Environments.

Environment	Location of Information	Open Source	FreeWare
Image2000	http://invision.gsfc.nasa.gov/image2000/	Yes	Yes
ImageJ	http://rsb.info.nih.gov/ij/	Yes	Yes
Java3D	http://java.sun.com/products/java-media/3D/index.html	Yes	Yes
OpenGL	http://www.opengl.org	No	Yes
Pixies	http://www.apteryx.fr/en/pixies/	No	No, evaluation version available for free

2.2.1 Evaluation of OpenGL

OpenGL is an environment for developing portable, interactive 2D and 3D graphics applications. It was introduced in 1992. It is supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC.

Advantages:

- Widely used in imaging applications. Many would even say that it is the industry standard in 2D and 3D graphics programming.
- Extensive functionality and ability to deal with new hardware.
- Platform independence.
- Well documented. Tutorials and sample code are readily available.
- Availability of discussion groups

Disadvantages

- Large learning curve due to extensive functionality. One has to be able to narrow down the focus to the task at hand. This may be difficult, as one may not know which functionality of OpenGL will be valuable for their task.
- Programs are typically written in C and C++. To use Java one would need to install a Java OpenGL binding, such as GL4Java. If one's language preference is Java, one may find that the documentation for OpenGL in Java is not as extensive as in C or C++. There may also be more roadblocks in finding how to do certain tasks in OpenGL with Java.

2.2.2 Evaluation of Java3D

Java3D is Sun's answer to a 3D API in the Java programming language. It provides a set of object-oriented interfaces that support a simple, high-level programming model. This gives developers the ability to build, render, and control the behavior of 3D objects and visual environments. At the low level, it uses DirectX or OpenGL to take advantage of 3D hardware acceleration.

Advantages

- Java's web compatibility with applets
- Open source code which allows programmers to get a deeper understanding of the objects at hand and customize the code to suit their needs.
- Java3D provides online documentation libraries and tutorials similar to that of the Java SDK documentation. It has a 3D programming structure that is explained very well in the online tutorials. The programming structure examples involve creating scene graphs and then programming code based on the scene graphs. Scene graphs are much like flow charts, but are specifically designed to explain a 3D environment.

Disadvantages

- From the information studied it seems that Java3D was designed to create 3D graphics at a less detailed scale than individual pixels. For medical imaging a more detailed approach is required.
- Large learning curve due to extensive functionality and new scene graph concepts. Although Java is platform independent Java3D is not just yet. It runs on UNIX and Windows, but does not yet support operation on Mac's.

2.2.3 Evaluation of ImageJ

As noted above ImageJ is a platform independent image processing application written in the Java programming language and developed by NIH.

Advantages

- Java's platform independence.
- Java's web compatibility with applets.
- Open source access
- Extensive documentation libraries with online tutorials and discussion groups.
- Focused on manipulation of captured images.
- Smaller API than OpenGL and Java3D therefore learning curve is also smaller.

Disadvantages

- No readily available 3D environment.

2.3 Chapter Summary

After an evaluation of the imaging environments above, the ImageJ application was chosen to be the best suited for the project's requirements. The ImageJ application was chosen because it allows programs to be written using the Java programming language and comes included with many image processing functions.

Chapter 3: 2D Manipulation of Images

ImageJ has a readily available set of two dimensional manipulation tools for use. To extend the two dimensional capabilities of Image J to suit the needs of the project a GUI was developed with a set of functions that were present in ImageJ. A new plugin ,Combine Stacks, was also developed to allow six image stacks to be combined into a single image stack which could then be animated. The motivation behind the development of these two parts was to provide a user friendly interface for manipulation of the spherical image stacks, allow the user to view the development of the embryo (spherical object of interest) from all six sides of the two dimensional picture stacks in a single image window and also provided the authors with an introduction on image processing in Java using the ImageJ standard libraries.

3.1 2D Graphical User Interface

The two dimensional graphical user interface has several functions that are found standard in the ImageJ application. Functions that were seen to be most useful were added to the GUI. The GUI layout can be seen in Figure 3-1.

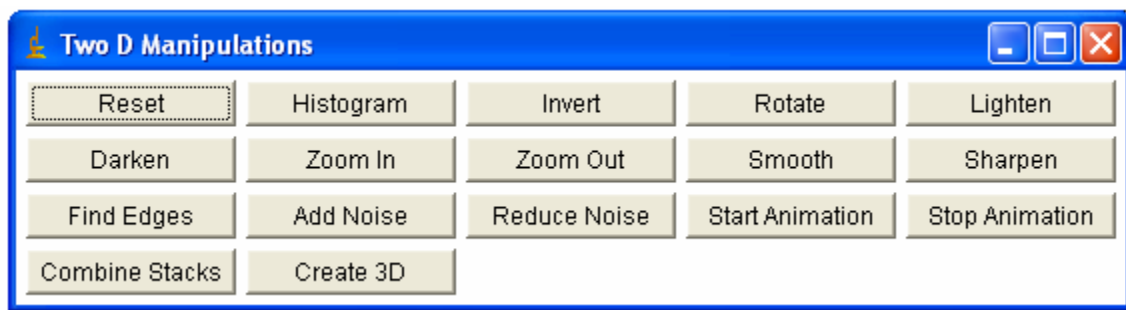


Figure 3-1. Two Dimensional GUI.

3.1.1 Functions

A description of each of the buttons pictured in Figure 3-1 follows:

- Reset – Provides the ability to return the image to its original state.
- Histogram – Returns the histogram for the selected image window.
- Invert – Inverts the image for active window
- Rotate – Rotates the object in 90 degree intervals.
- Lighten/Darken – Multiplies each pixel within a 10 percent range
- Zoom in/out – Scales the image by 20%
- Smooth – Replaces each pixel with the 3x3 neighborhood mean.
- Sharpen – Sharpens the image or ROI (Region of Interest) using a 3x3 convolution kernel
- Find Edges – Finds edges in the image or ROI using a Sobel operator.
- Add /Reduce Noise – Adds random noise to the image or ROI.

- Start/Stop Animation – Starts and stops animation of stacks.
- Combine Stacks – Combines six separate image stack windows into one image window.
- Create 3D – Create a 3D image from a set of 2D images.

3.2 Combine Stacks Plugin

The Combine Stacks plugin is a new function that was implemented to allow users to view six separate image stacks in one image window. The image stack could then be animated to produce a movie. This function is intended to allow users to study the development of the spherical object, e.g. a living embryo, from all six sides of the object through time.

3.2.1 Combine Stacks Algorithm

The Combine Stacks plugin uses some of the ideas that were found under the ImageJ plugin, "Stack Combiner" written by Wayne Rasband². The following is a description of the algorithm used to implement the Combine Stacks function with an accompanying flow chart as shown in Figure 3-2.

- Open six image stacks.
- Check if six image stacks are open and that they all have the same number of slices.
- Check if all filenames are unique and follow the naming convention <filename>_top, <filename>_bottom, <filename>_left..., etc.
- Get the maximum width and height of the image stacks for the case of unequal image dimensions
- Create a new window with the dimensions 3 x width and 4 x length.
- Identify each side of the image according to file name convention and place into the appropriate position as shown in Figure 3-3 and Figure 3-4 shows the result of combining six image stacks using the Combine Stacks plugin.

² Rasband Wayne (wayne@codon.nih.gov) .2001.<http://rsbweb.nih.gov/ij/plugins/combiner.html>

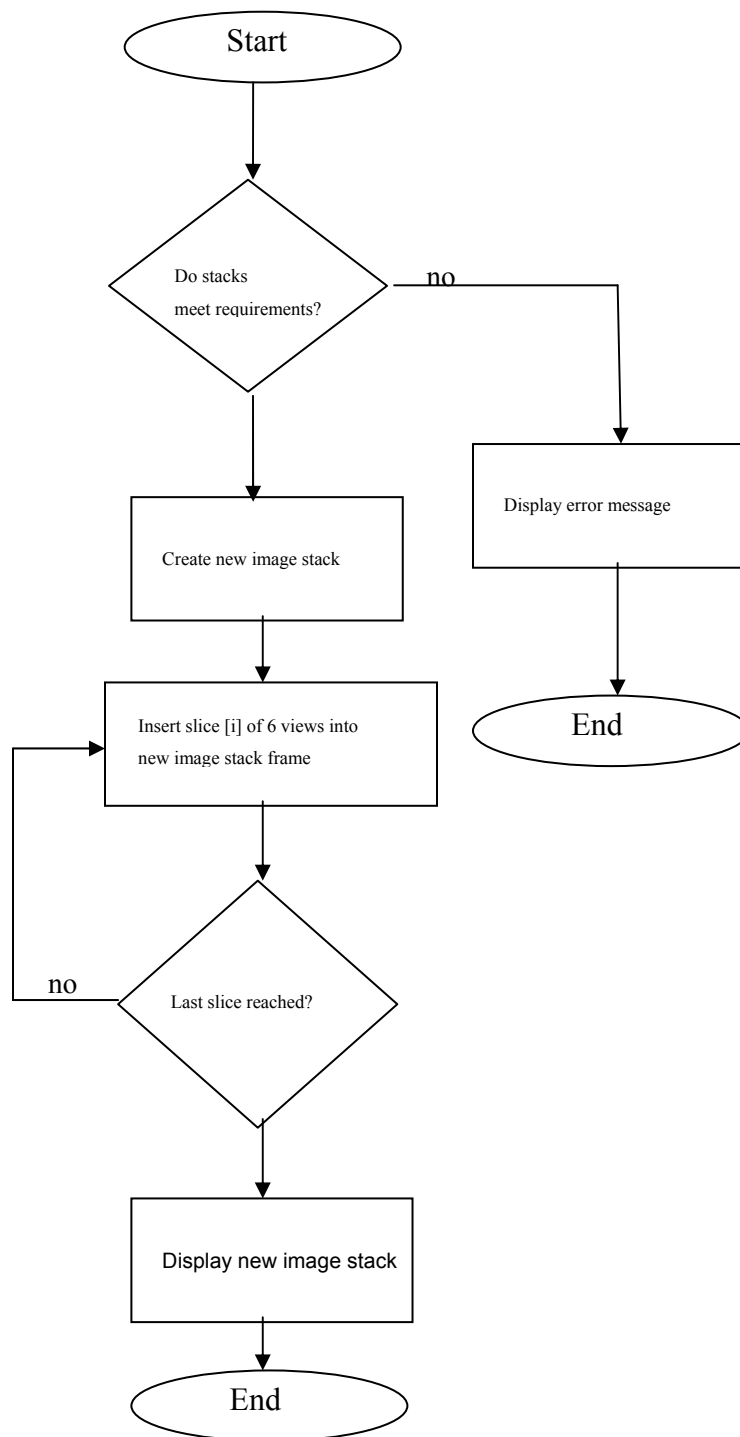


Figure 3-2. Flow chart of Combine Stacks plugin

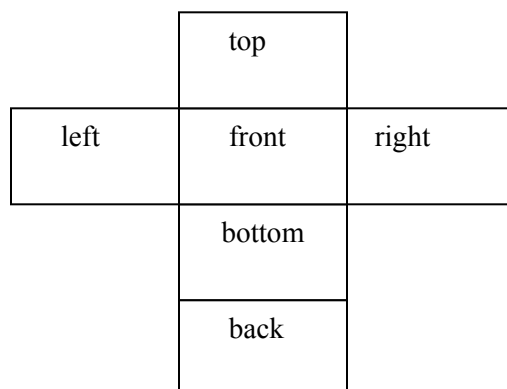


Figure 3-3. Placement of pictures.

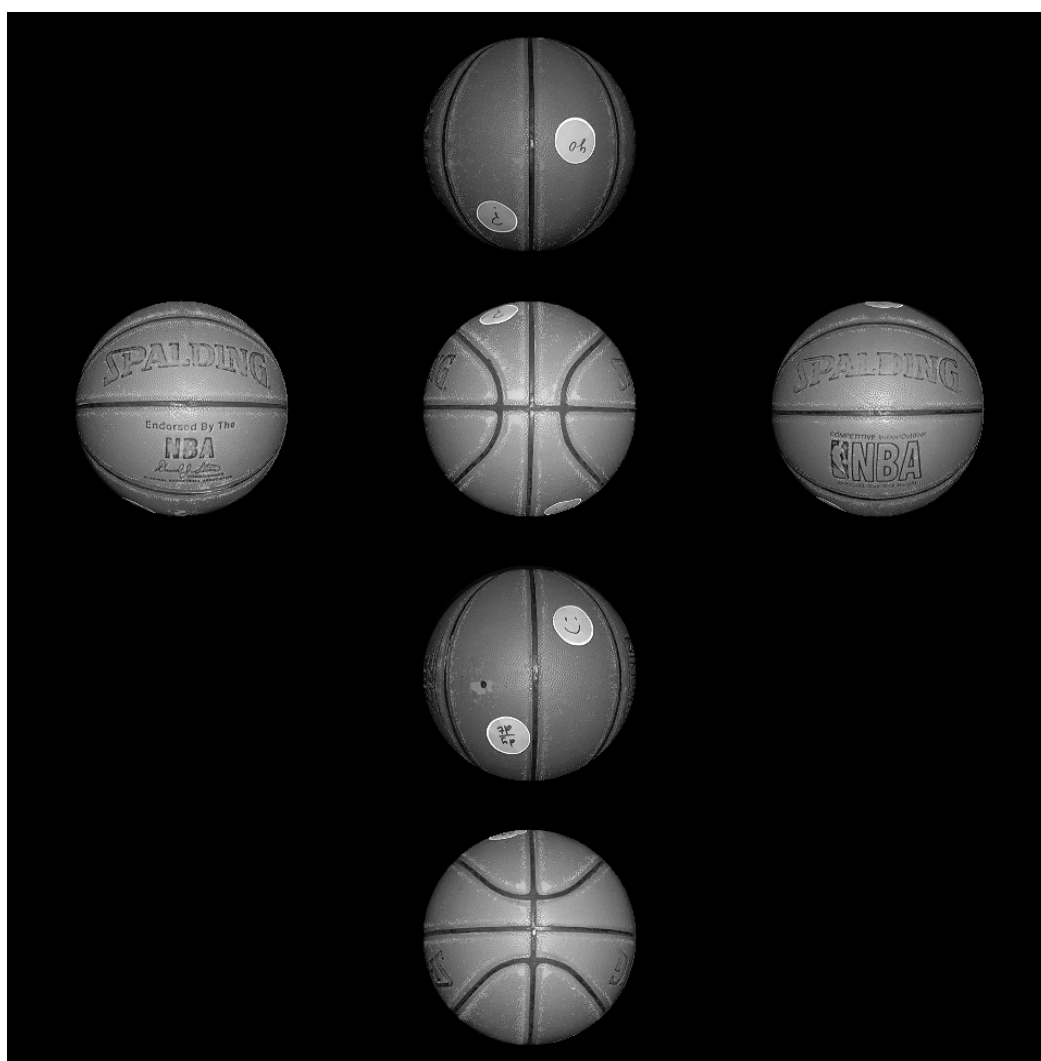


Figure 3-4. Combined image stacks of a basketball.

3.2.2 Proper Use and Limitations

The Combine Stacks plugin requires that certain requirements be met in order for the plugin to work properly. The following is a list of requirements for proper use and limitations of the Combine Stacks plugin.

- Six stacks of equal stack size must be used.
- Stack images must be named according to the following convention <filename>_top, _bottom, _left, _right, _front, _back.
- Stack images should be of the same length and width (optional).
- Two dimensional manipulation of the combined image stack will affect all six images.

3.3 Chapter Summary

The work performed in the 2D manipulation section resulted in a GUI created with several ImageJ functions. A plugin was also developed to display six 2D images simultaneously. Furthermore, the tasks in this section helped the authors gain a good understanding of software development using the Java language and the imaging functions that are provided by the ImageJ application software.

Chapter 4: 3D Creation of Spherical Objects

Software was developed to create a three dimensional image object from six 2D images of a spherical object. Each of the six input images corresponds to one side of a cube. The six 2D images were mapped onto a sphere to create the 3D spherical object. The creation of the 3D image can be broken down into seven steps. This software was developed as a Java plugin for ImageJ. The Java code for this program can be found in Appendix B. The methodology behind the 3D imaging section is discussed below.

4.1 3D Software Assumptions

Some of the assumptions the program makes are:

- The six images are perpendicular to each other.
- The object to be reconstructed is perfectly spherical.
- The background is within a certain range of greyscale values.
- The lighting of the object is constant around the entire sphere.
- All six images are of the same magnification.

4.2 Step 1: Load Images

This step involves creating an image processor for each of the six 2D images and loading the image data into each of the image processors. Each of the six 2D input images corresponds to one side of a cube. To get a better understanding of how the 2D images look please see the six pictures taken of the sample basketball, Figure 3-4 on page 9.

4.3 Step 2: Center Images

This step takes each of the six images and centers the spherical object with the center of the image frame. The basic idea here is to find the center of the sphere in 2D space and use this to center the sphere with the image frame. The method sets a greyscale threshold range, which corresponds to the background of the image. Any pixels outside the threshold would correspond to the spherical object. The average of the pixels is taken in both dimensions. The average corresponds to the center of the sphere in the 2D view. The steps in the method to center the image horizontally are listed below.

- First, the program sums the pixel x location of all pixels outside the background range in each row. A count is kept of the valid number of pixels; a valid pixel is one which is outside the background range. A count is also kept of the max number of consecutive valid pixels, $c_{validmax}$.
- If $c_{validmax} > n$ then the row is valid since it contains at least n consecutive valid pixels. In our program n was arbitrarily chosen to be 10.

- If the row is valid the average center of the row is calculated. Then we move onto the next step.
- If the row is not valid then we move onto the next step.
- The program checks to see if all rows have been completed. If not all rows are completed the program goes back to the first step of this method. If all rows are completed the program takes the average center of each row, and calculates the average x center of the object.
- Each row is shifted to have the center of object same as the center of the image frame.

A similar algorithm can be used to center an object vertically.

4.3.1 Limitations

There are limitations to this method of centering the object. The main limitation is due to the assumption that the background range will be in a certain range of greyscale values. This implies that any pixel in the object within the background image is not considered to center the object. This assumption is unacceptable if the background pixels within the object are not evenly distributed, as this will result in the image being off center.

4.3.2 An Alternative Centering Approach

Another option for centering the image would be to find the edges of the object to center it. The find edges function is available in ImageJ. This function uses fast Fourier transforms to find the edges in the image. A problem with the finding edges method is that it'll find all edges, including the shadow of an image, and any edges on the object. The program has to be able to distinguish which edge's are of the perimeter of the object from ones which are not. If this can be done, this method may prove to be a superior method in centering the image. For future work, one may consider using both methods to create an improved method to center the image.

4.4 Step 3: Find Radius of Each Image

This step takes each of the six images and finds the radius for each object. It is assumed that the object is centered from the previous step so that the object center is at the midpoint of the picture vertically and horizontally. In order to find the radius of each image, an estimate of the radius must be entered by the user. This is a simple procedure where the user simply finds the coordinate value of the edge of the object and subtracts that from the corresponding midpoint value. The program will then use this information to find the radius by taking an average of all the radii obtained for every angle in one revolution. Each angle is incremented 0.01 radians. For each radius value at a particular angle, the pixel values must be obtained from the midpoint

coordinates to the end coordinates of the edge of the object. ImageJ has a built in function, `getLine(x1,y1,x2,y2)`, which returns an array containing the pixel values along the line starting at (x_1,y_1) and ending at (x_2,y_2) . To find x_2 and y_2 values for a particular angle the following formulas are used:

$$\begin{aligned}x_2 &= x_1 + r \cos(\theta) \\y_2 &= y_1 + r \sin(\theta)\end{aligned}\tag{4.1}$$

where $0 \leq \theta \leq 2\pi$

Once the array of pixel values is obtained each value is compared with the threshold value. The threshold value is a value that is chosen within the range of the background pixel values. If the pixel value for each array element is within the threshold, it is assumed to be the end point of the radius. This will continue until the end of the array or the consecutive number of threshold values is greater than a specified number. To eliminate inconsistencies radii that are smaller than a given value are not counted. The resulting radius is the average of all the radii for one revolution.

4.5 Step 4: Calculate Third Coordinate

This step takes each of the six images and calculates the corresponding third coordinate. It is possible to calculate the third coordinate when the shape of the image object is known. If the shape is unknown it is mathematically impossible to define the shape of the object³. For this project, however, the image objects under observation will always be assumed spherical. From the 2D images that we obtained from previous steps, we know two of the coordinates for the sphere therefore the third coordinate can be calculated with the equation for a sphere, see Equation 4.2. In Equation 4.2 the image is centered at $(x,y,z) = (r,r,r)$, where r is the radius of the sphere.

$$(x - r)^2 + (y - r)^2 + (z - r)^2 = r^2\tag{4.2}$$

³ Dr. Derek Meek, Dept. of Computer Science, University of Manitoba, dmeek@cs.umanitoba.ca, personal communication.

Since the equation is quadratic there will be two possible answers for the third coordinate. Table 4-1 shows the range of the third coordinate from different views. These ranges are used to choose the correct value for the third coordinate. To have a better understanding of the images and their coordinates see Table 4-1 and Figure 4-1.

Table 4-1. Third Coordinate Value

View	Coordinates Available	Third Coordinate Value
Front	x, y	$r \leq z \leq 2r$
Back	x, y	$0 \leq z \leq r$
Bottom	x, z	$0 \leq y \leq r$
Top	x, z	$r \leq y \leq 2r$
Left	y, z	$0 \leq x \leq r$
Right	y, z	$r \leq x \leq 2r$

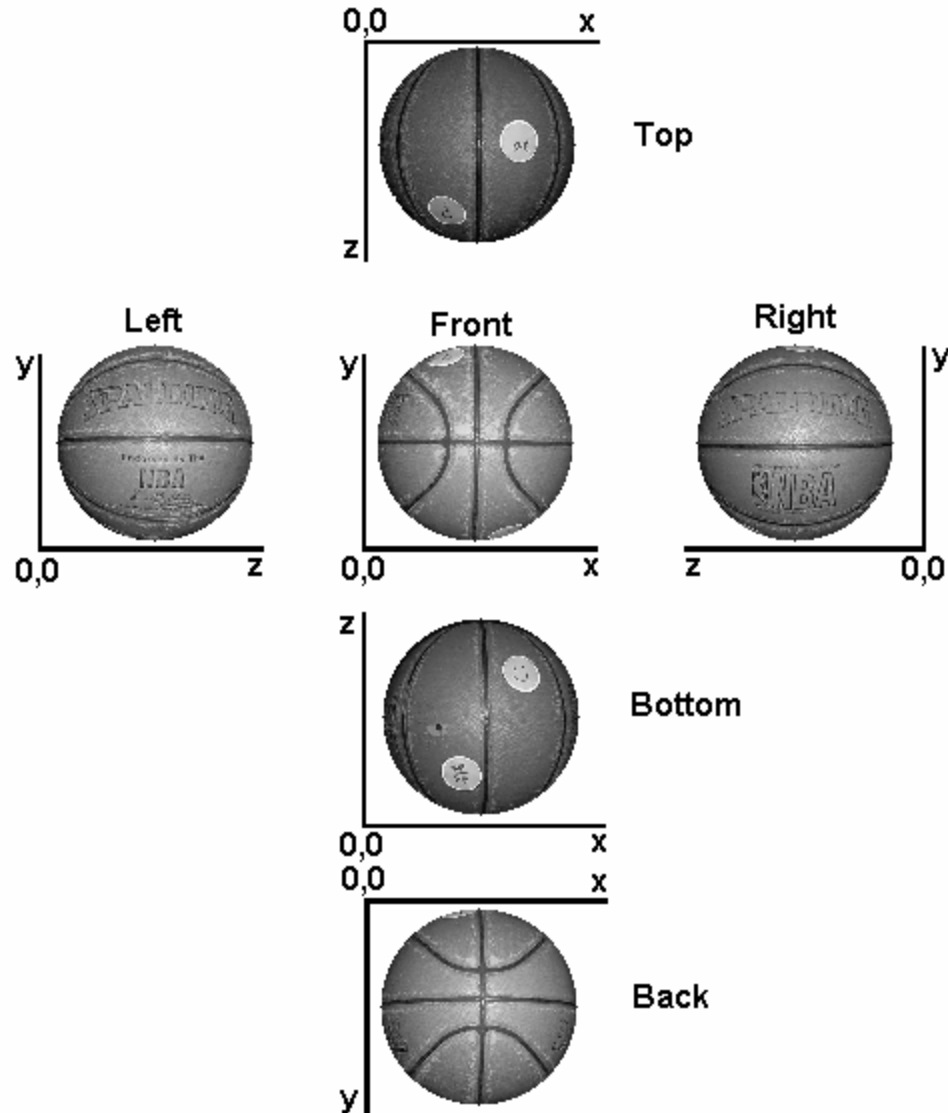


Figure 4-1. Coordinate Structure of Images.

After the third coordinate is calculated, the x , y , z coordinates are stored for each view along with the appropriate greyscale value. This data is then passed onto Step 5 Calculating Weighted Greyscale.

4.6 Step 5: Calculate Weighted Greyscale

From the previous step, the program is given the x , y , z coordinates of all the voxels. A voxel is a volumizing element in 3D space, corresponding to a pixel in 2D space. A voxel can be thought of as the smallest cube shaped division of 3D space. Each of the voxels requires a greyscale value to display. This step shows how the greyscale value is calculated for each voxel.

Most voxels will be present in three of the six views. Some may be present in two, four or five of the views. There will be a maximum of up to six voxels which are present in five views. These would be the voxels that are ninety degrees to one of the six planes of view.

Each view may have a slightly different greyscale value. The program has to decide which greyscale value should be used for each voxel. For weights we use the squares of the three direction cosines of radii of the sphere, for which:

$$\cos^2\alpha_x + \cos^2\alpha_y + \cos^2\alpha_z = 1 \quad (4.2)$$

where the angles are measured from the three principle axes. When only two views project onto the object, the third angle is ninety degrees. When four views project onto the object at minimum one of the angles is ninety degrees. When five views project onto the object at minimum two of the angles are ninety degrees. The weights of all ninety degree angles are zero, resulting in a maximum of three projections with a weight on any voxel.

From direction cosines of radii of the sphere, the weight of each plane is determined. The greyscale value of the voxel can be calculated as follows:

$$V(x, y, z) = \sum_m g_m \cos^2 \alpha_m \quad (4.3)$$

Where $V(x, y, z)$ = the greyscale value of voxel at x, y, z
 $m = 1, 2, 3, 4, 5$ or 6 corresponding to one of the six views and representing the two to five views projecting onto a voxel. A maximum of three views will have non ninety degree angles.
 g_m = greyscale value of coordinate in plane m
 α_m = angle between plane m and normal to surface of sphere
and $-90^\circ \leq \alpha_m \leq 90^\circ$
0 is black in greyscale and 255 is white in greyscale.

From this equation the new greyscale value is calculated and stored. The original views and their weighted greyscale counterparts can be seen in Figure 4-2 and Figure 4-3 respectively.

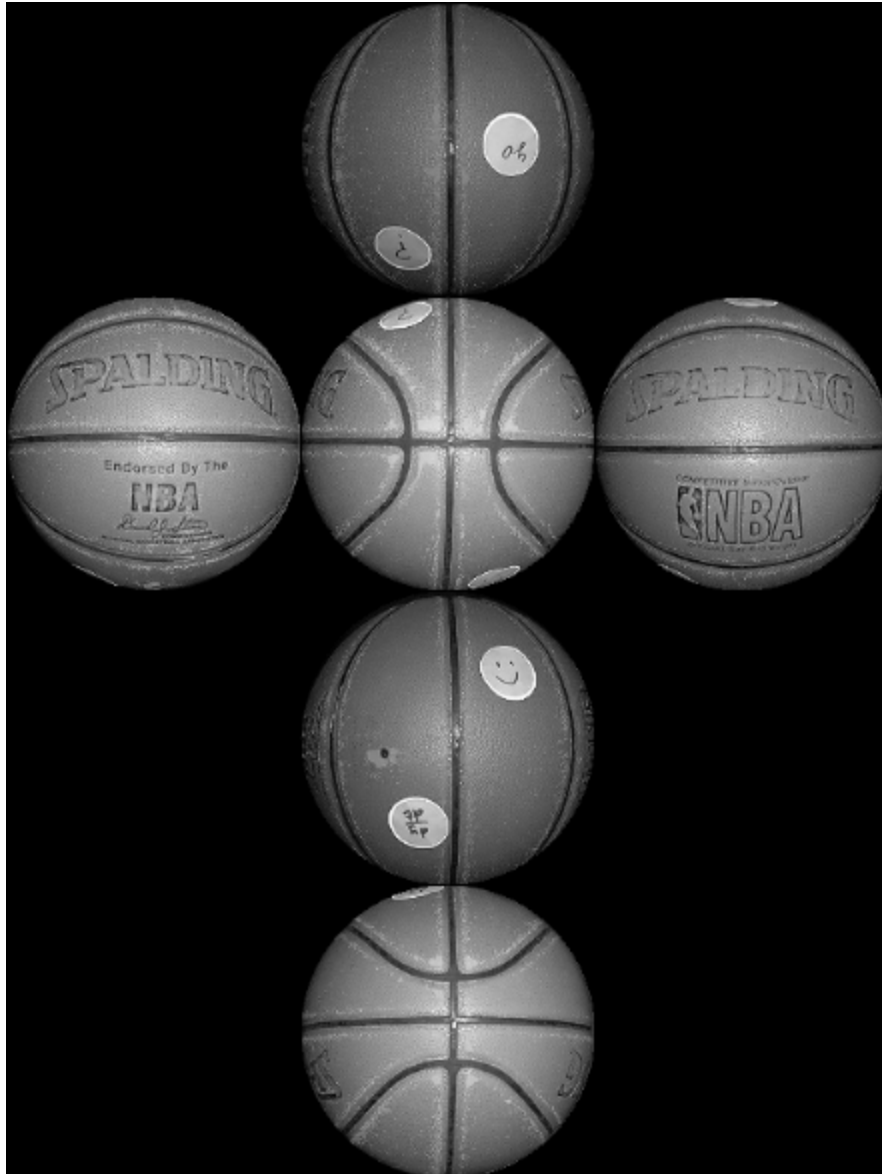


Figure 4-2. Original Views.

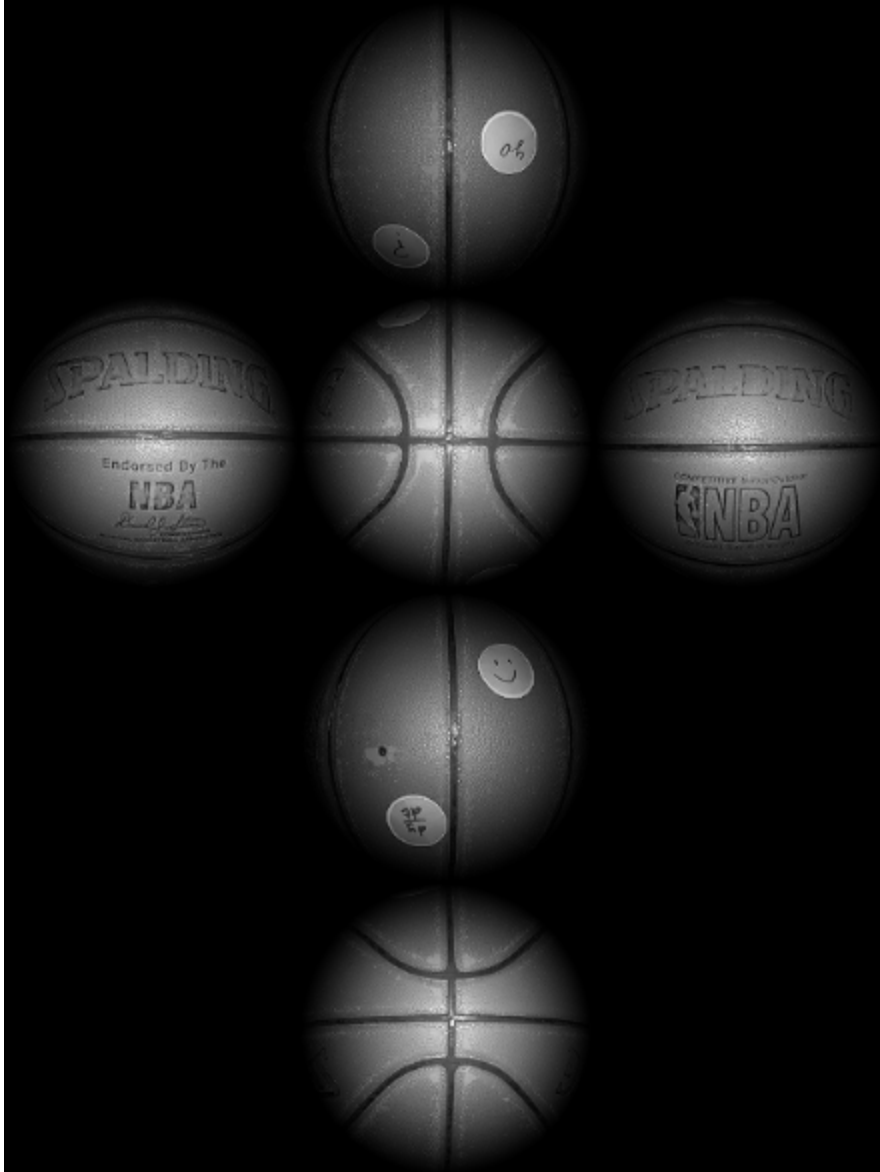


Figure 4-3. Views with Weighted Greyscale.

Notice in Figure 4-3 that the images fade at the edges of the circle. Each projected view becomes less and less significant as it moves away from the center. The fading of pixels in one view corresponds to unfaded pixels in corresponding views. For example, the front view fades away at its pixels on the right side. In the right view these same pixels are in the center of the image and are not faded at all. This weighted greyscale scheme allows for the view with the most direct view of the pixel (voxel) to have the most significance on its greyscale value.

4.7 Step 6: Display each of the Six Images with new Greyscale

The simplest form of displaying the images would be to redisplay the six views, with the recalculated voxel values. The before and after pictures of the six views can be seen in Figure 4-2 and Figure 4-4 respectively. Further analysis of the results can be found in section 5.1 and 5.2.

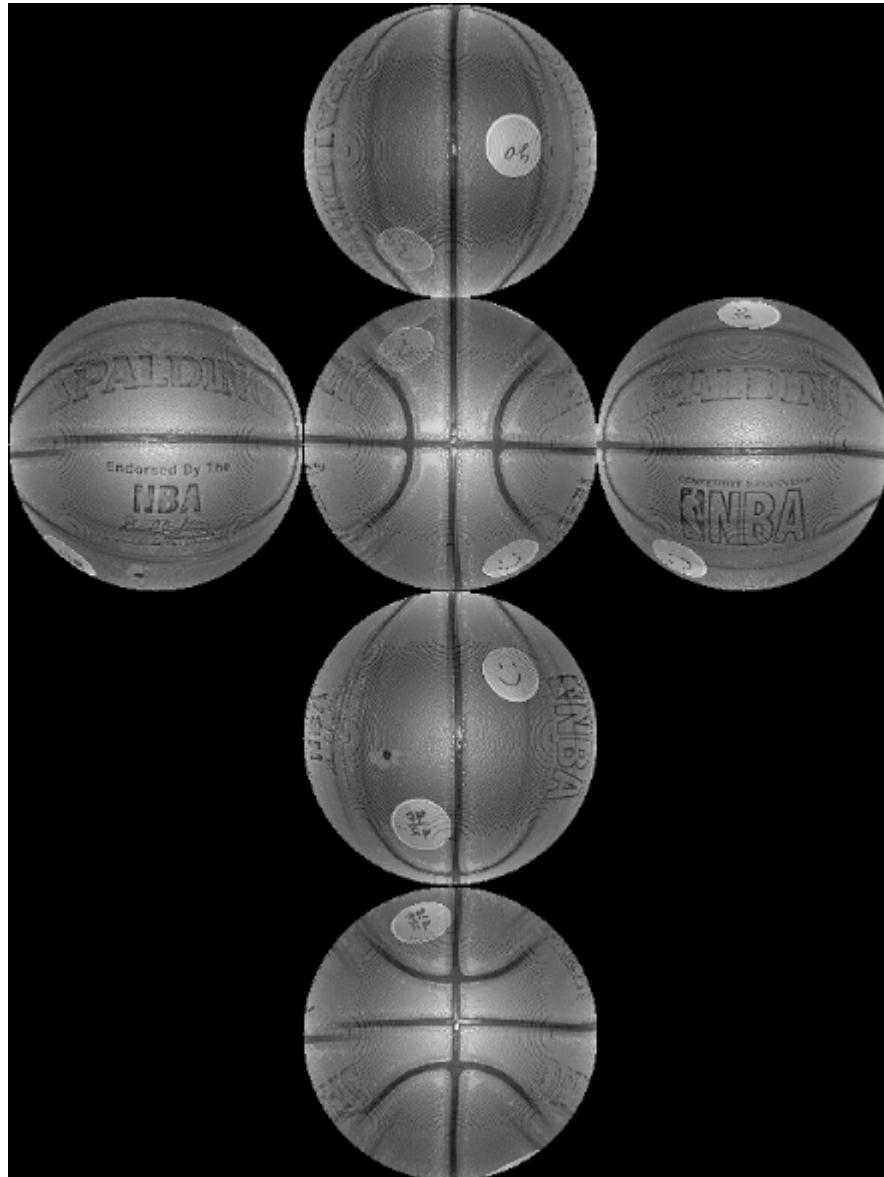


Figure 4-4. Views Reconstructed from Weighted Greyscale views.

4.8 Step 7 Rotate Image Object

Due to time constraints this step was not completed. Please see section 5.4 for a discussion on the theories behind 3D rotation.

4.9 Chapter Summary

The work performed in the 3D creation section resulted in the creation of a 3D spherical object from six different 2D views. Several functions had to be developed to reach this goal.

These functions included:

- Loading images
- Centering images
- Finding the radius of each image
- Calculating the third coordinate of each image
- Calculating the greyscale values of 3D image
- Displaying each of the six images with new greyscale values

This Java encoded software can be used as a plugin in ImageJ.

Chapter 5: Future Work

There is a significant amount of work that can be done to improve the functionality of the 3D imaging program. This includes:

- Improving the calculation of greyscale value algorithm
- Calibrating the images to be perpendicular
- Reducing computing time
- Implementing a 3D rotation method.
- Considering the use of commercial software

5.1 Improving Calculation of Greyscale

Taking a close look at the reconstructed views in Figure 4-4, one will notice that there are some ring artifacts. This may be due to an attribute of circular/spherical images not considered when calculating the weighted greyscale value. These artifacts may be caused when only two views hit the voxel, and the third view is blocked by another voxel. An example of this is shown in Figure 5-1 below, where the pixel labeled would be blocked by the pixel to its right from being projected onto by the view to its right. Thus the calculated greyscale is missing its third weight and is darker than it should be. To solve this problem one could count the number of views projected onto each voxel. If only one or two views project onto a voxel its greyscale can be calculated with a different method. For only one view projecting onto a voxel, the greyscale will be the same as the projecting view. If two views are projected onto a voxel the greyscale can be calculated as a percentage of the sum of the $\cos^2\alpha_m$ of both views. If there are three or more views projecting onto a voxel the greyscale can be calculated as described in Section 4.6 Step 5: Calculate Weighted Greyscale. Note that there is a difference when only two views are projected onto a voxel and when only two views are used to calculate the greyscale when the third view is projecting at ninety degrees. The former would use the method mentioned in this paragraph to calculate the greyscale and the latter would use the method in section 4.6.

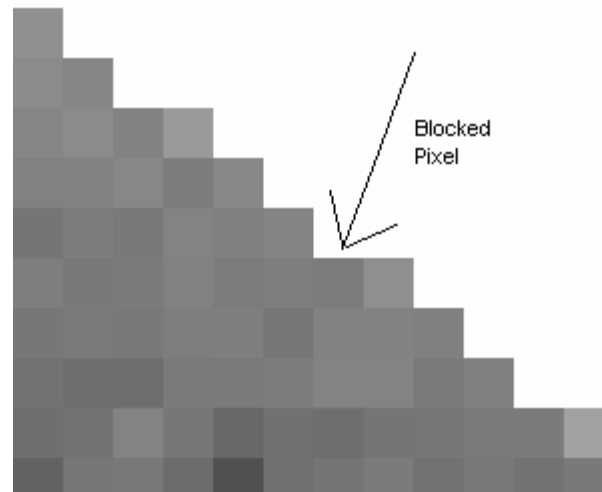


Figure 5-1. Blocked Pixel on Sphere

5.2 Calibrating Images to be Perpendicular

Another noticeable difference between the original views and the views reconstructed from the six weighted greyscale images is that some of the features of the ball do not line up. This is evident in all six views. A reason for this may be that the images are not perfectly perpendicular to each other. This can be seen when taking a close look at Figure 4-2, for example the vertical stripe in the center of the front view does not line up with the vertical stripe down the center of the bottom view. To simplify the design the program developed assumed that the views will be perfectly perpendicular to one another. With real images this may not always be possible. To fix this a future design should consider calibrating the images. After calculating the third coordinate for a view the image could be rotated to make the views perpendicular to one another. For this to occur the rotation function discussed in section 5.4 will have to be implemented. Each view will have to be calibrated before the weighted greyscale value is calculated. The degree to rotate by will have to be calculated manually and entered into the program, as it will be extremely difficult to give the software intelligence to complete this task. To determine the degree of rotation trial and error may have to be used, as it may also be difficult to determine with the human eye. In the case of a stack the angle of rotation will have to be determined only once per view, as all stack images will require the same angle of rotation.

5.3 Reducing Computing Time

Computing the 3D voxels is very time consuming. The majority of computation time involves calculating the weighted greyscales. For a sphere with a radius of 98 pixels, each view

has 30147 pixels on the sphere. The program was run on three different computers. The processing time can be seen in Table 5-1 below.

Table 5-1. Processing Time.

Processor	Ram	OS	Computational Time
AMD K6 300MHz	160MB	Windows 2000	44.47 minutes
PIII 850MHz	128MB	Windows XP	22.18 minutes
AMD Duron 850MHz	256MB	Windows XP	17.61 minutes

Table 5-2 below shows the time it takes to process each view on the AMD Duron 850MHz processor. For the remaining processors data is available in Appendix C.

Table 5-2 Processing Time per View

View	Number of Voxels	Time (minutes) to process current view
0	30147	1.00
1	60292	3.00
2	72796	3.09
3	85301	3.36
4	92801	3.11
5	100302	4.04
	Total Time	17.61

As seen in Table 5-2 the processing time per view increases as more views are processed. The processing time of view one is significantly lower as no processing time is spent in searching to see if the voxel coordinates are present in the array. This is because the array is empty before processing of view one begins. For the rest of the views, most of the processing time is spent searching to see if a voxel coordinate is present in the array. One way to possibly reduce this time is to order the array after a view has been processed. This will allow one to create an efficient search algorithm to see if the current voxel coordinate is present in the array, if not the voxel coordinate is added to the end of the array. Since all voxel coordinates for a view will be unique within that view, only coordinates from previous views will need to be searched. This may significantly reduce the processing times of views one through five.

5.4 3D Rotation

With regards to time, the rotation of the 3D object could not be implemented fully. However, background research has been done to explore some ideas that are used to rotate an object in 3D space [see 5].

5.4.1 3D Representations

In three dimensional space an object can be represented by a set of vectors $[x \ y \ z]$. This representation, however, is not sufficient when three dimensional transformations are required. This is due to the fact that transformation operations about the origin ($[0 \ 0 \ 0]$) are not effective. To overcome this, it is necessary to translate the origin and any other point in three dimensional space. This can be accomplished by homogenous coordinates. For three dimensional space the resulting four dimensional position vector will be of the form

$$[x' \ y' \ z' \ h'] = [x \ y \ z \ 1][T] \quad (6.1)$$

where h is a real number and T is a transformation matrix. The transformation matrix is a 4x4 matrix of the following form

$$[T] = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & i & j & r \\ l & m & n & s \end{bmatrix} \quad (6.2)$$

where each section of the matrix has an effect on the object. The 3x3 matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

for example performs a linear transformation, vector $[l \ m \ n]$ produces translation, $[p \ q \ r]$ produces a perspective transformation and s is involved in overall scaling. A combination of these produces the desired transformation such as rotations, reflections, scaling, etc., from any arbitrary point. Finally, to switch from homogeneous coordinates to ordinary coordinates the operation is given by

$$[x^* \ y^* \ z^* \ 1] = \left[\frac{x'}{h} \ \frac{y'}{h} \ \frac{z'}{h} \ 1 \right] \quad (6.3)$$

5.4.2 Rotation about the Coordinate Axis

From our discussion above, the fundamentals of 3D transformation can be extended to our need for rotation about the coordinate axes. When rotating about a coordinate axis the coordinates of that axis position vector do not change, which suggests that the rotation occurs in a plane perpendicular to that axis. For example, when rotating about the x -axis, the x coordinates of the position vectors do not change. This is the same for the y and z axes.

So in order to rotate about a coordinate axis for a specified angle all that is needed would be the position vectors of the object and the following transformation matrix will be required to rotate along that axis. For the x axis the transformation matrix would be

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.5)$$

the z axis transformation would be

$$[T] = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.5)$$

and for the y axis transformation would be

$$[T] = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.6)$$

Rotational angles are measured positive in the right hand sense. It should also be noted that since matrix multiplication is generally non commutative, the order in which rotations occur can affect the final resulting orientation of the object.

With this knowledge an initial algorithm has been developed for implementation of this rotation.

1. Receive user input of where to rotate (angle, x , y , or z axis)
2. Calculate transformation matrix from input
3. Obtain homogenous coordinates of the points
4. Matrix multiply to obtain new position
5. Display new image

This may be the only type of rotation that is required if we are interested in rotation about an axis. However, we may want the user be able to rotate the object from an arbitrary point and will require the transformation matrix to be manipulated.

5.4.3 Rotation about an Arbitrary Point

Rotation about an arbitrary point requires that multiple transformations be used in order to accomplish the task. A complete discussion of the mathematics involved can be found in *Rogers & Adams (1990)*. Presented below is an algorithm that is required to rotate about an arbitrary axis in space by some angle δ [see 5].

1. Translate so that the point (x_0, y_0, z_0) is at the origin of the coordinate system.
2. Perform appropriate rotations to make the axis of rotation coincident with the z -axis.
3. Rotate about the z -axis by the angle δ .
4. Perform the inverse of the combined rotation transformation
5. Perform the inverse of translation.

5.4.4 Section Summary

The rotational aspect of the 3D Imaging project could not be implemented. This is mainly due to time constraints. Rotational algorithms are readily available, which can be written in any programming language to allow for rotation of 3D objects.

5.5 Use of Commercial Software

Another area that can be explored further is commercial software that can be used to produce a realistic 3D spherical image object. Consulting with Terry Walker, Coordinator of Computing/CADLab Faculty of Architecture University of Manitoba, and Steve Shaw, a graduate student in the Faculty of Architecture, we received suggestions that two types of software may be

suitable to obtain 3D images of a sphere from six different views. The software packages suggested are FormZ and Cinema4D. Both programs are currently available at the Architecture computer labs at the University of Manitoba. Both applications are relatively inexpensive, under \$1000US, compared to imaging software that is currently available. Both applications are quite similar, with Cinema4D having additional functionality. It was suggested that Cinema4D may be suitable for our needs. Therefore only a discussion of Cinema4D is presented.

Cinema4D is an application that is widely used by many industries that require high resolution pictures and animation to be created. Industries ranging from entertainment to architecture and even the medical community all have embraced Cinema4D. This software has many of the features that would be appropriate for 3D realism. Some of the functions available are [see 2]

- Rendering tools that are capable of 256 bit rendering and anti-aliasing which gives smooth edges and sharper images. The software also allows multiprocessor rendering which can be useful when many high resolution images are needed.
- An extensive collection of camera views and an adaptive shading engine which can aid in modeling.
- A symmetry tool that will mirror a symmetrical object.
- Hypernurb technology which will morph standard shapes into new objects.
- Comprehensive animation tools capable of animating anything.
- Apply “materials” on objects which will give them a realistic look. 2D images can be scanned in and projected onto various shapes of objects.
- A powerful illumination system which can imitate any light source from any angle.
- The software supports many image formats and commercial file formats.

Some of these functions were presented to us by Steve Shaw. He displayed six images that were created in Adobe Photoshop which would be similar to the multiple views of a spherical object. He then introduced the concept of hypernurbs which are shapes that can morph into new shapes when placed near other objects. This allowed the six images to be projected onto a sphere. This seems like the result that is desired. One problem noticed is that the images overlapped each other, but only the last image projected is seen. It may be possible to get past this hurdle. There are user newsgroups available for Cinema4D to help discuss problems.

Cinema4D is a tool that has the potential to be very useful in the project being discussed. It has some limitations, including:

- A large learning curve to become proficient in Cinema4D. The program was designed for users with knowledge of imaging.
- Inability to control all pixels in an image individually.

5.5.1 Section Summary

Cinema4D is a powerful tool that can model objects with great detail. The functions available offer a variety of choices for the user. However, it may be difficult to become proficient with the use of the software for beginners and may not have the desired accuracy.

5.6 Chapter Summary

This chapter discussed possible future work that can be done to improve the functionality of the 3D imaging program. The 3D imaging software developed can be used as a stepping stone by future researchers in creating fully functional 3D imaging software for spherical objects.

Chapter 6: Application to 3D Spherical Objects

The 3D imaging project is a small part of a larger project called the 4D embryo imaging project. The goal of the larger project is to take images of growing salamander embryos and produce 3D images over time. By having a set of 3D images produced scientists may gain an understanding of how an embryo builds itself, and what goes wrong when birth defects occur. The end goal of the 4D embryo imaging project is to observe, for the first time, the common human birth defects (also common in salamanders) of spina bifida and anencephaly (neural tube defects) in salamander embryos. Understanding of these ideas may lead to prevention and possibly even cure in humans.

6.1 Procedure

The project requires acquiring images of growing salamander embryos. To acquire the images data acquisition will have to be automated. A system has been proposed that will take pictures of embryos of a diameter of 2mm. The embryo will be sitting on a glass cover slip, of standard 0.17mm thickness. Four 3.2 mm tall glass prisms will surround it. This will allow one to take 5 images through the bottom of the glass slide. Another microscope/camera from the top will retrieve the top image as seen in Figure 5-1. This whole system will be under water during the whole process. The system will be kept at room temperature. If needed the system temperature can be cooled to slow down the embryo development, or raised to speed it up. Outside of the normal temperature range of 8-24 deg C, abnormalities tend to occur.

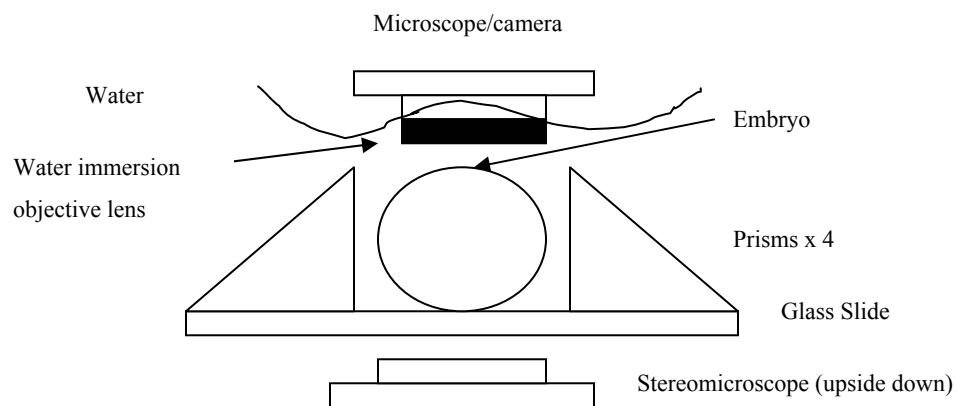


Figure 6-1. Front view of the proposed system.

One aspect to be decided is how many pictures need to be taken. This decision depends on the use of the optic lens in acquiring images. It may be possible to take a single digital picture

and focus the image later through computer aided manipulations [see 4]. If this can work on such small geometries only two images will be required, one from the top and one from the bottom (which will include the four side images with the prisms). If this hypothesis about the optic lens can not work on small geometries, multiple images of each of the six sides will need to be taken. This would allow several layers of focused parts to be combined together [see 6]. Alternatively, an optical approach may suffice: by gluing a square piece of glass under the coverslip, the optical path length for the image seen straight through the bottom could be increased to match the optical path length of the images going through the prisms.

After data acquisition is automated the images may need to be filtered and manipulated before displaying. Software is needed to center the images and calculations will need to be made to take the two dimensional images and merge them to a three dimensional “life like” representation of the embryo. This is what the software developed for this project will be used to do.

6.2 Chapter Summary

The 3D imaging project is a significant part of a larger project, the 4D embryo imaging project. The larger project requires that images be acquired of embryos by the methods described previously. After the acquisition of the images, the software that was designed in the 3D imaging project can be used to create a life-like representation of the embryo.

Chapter 7: Conclusion

From the work that was done on this design project, a platform independent software application was developed to manipulate 2D images. A GUI was created that allowed for manipulation of 2D images, including the displaying of six consecutive images over time. A 3D spherical object was created from six 2D images. This was developed in a cost effective way with the use of ImageJ and the Java API's. To improve the functionality of the 3D software future suggested work includes optimizing the calculation of greyscale algorithm, calibrating images to become perpendicular, reducing computing time, rotating the spherical object, and using third party software to create a realistic 3D spherical object.

Appendix A: 2D Imaging Java Code

See attached file “Two_D_Manipulation.java” and “Combined_Stacks.java”

Appendix B: 3D Creation of Spherical Object - Java Code

See attached file "Run_3D_.java"

Appendix C: Computation Time Tables

Below are the tables for the computation time required to process each of the six views of the 3D image on different processors.

Table C-1. Processing Time per View on AMD Duron 850MHz

View	Number of Voxels	Time (minutes) to process current view
0	30147	1.00
1	60292	3.00
2	72796	3.09
3	85301	3.36
4	92801	3.11
5	100302	4.04
	Total Time	17.61

Table C-2. Processing Time per View on PIII 850MHz

View	Number of Voxels	Time (minutes) to process current view
0	30147	1.27
1	60292	3.87
2	72796	3.83
3	85301	4.26
4	92801	3.87
5	100302	5.08
	Total Time	22.18

Table C-3. Processing Time per View on AMD K6 300MHz

View	Number of Voxels	Time (minutes) to process current view
0	30147	2.82
1	60292	8.78
2	72796	8.83
3	85301	9.84
4	92801	9.08

5	100302	11.78
	Total Time	44.47

References

- [1] Ammeraal, Leen.1998. *Computer Graphics for Java Programmers*. England: John Wiley & Sons.Rogers,
- [2] Bailer, Werner (werner.bailer@fhs-hagenberg.ac.at).2001."Writing ImageJ PlugIns – A Tutorial." Writing ImageJ PlugIns – A Tutorial. *Fachhochschule Hagenberg University*. Internet web site. Accessed 05 January 2002.
URL: <http://webster.fhs-hagenberg.ac.at/staff/burger/ImageJ/tutorial/>
- [3] Beichert, Dirk, Joachim Heller, and Onur Pekdemir.2001."Cinema 4D XL7 Release." *Maxon Computer* .Internet web site. MAXON Computer GmbH. Accessed 15 February 2002.URL: <http://www.maxon.net/usa/index.html>
- [4] DowskiÊ Jr., E.,and G. Johnson .2002. "Marrying optics & electronics. Aspheric optical components and electronics improve depth of field for imaging systems." *SPIE's OE Magazine*, January, 42-43.
- [5] David F., and J. Alan Adams.1990. Three Dimensional Transformations and Projections. Chap. 3 in *Mathematical Elements for Computer Graphics Second Edition*. New York: McGraw-Hill Publishing Company.
- [6] Pieper, R.J., and A. Korpel. 1983."Image processing for extended depth of field." *Appl. Optics* 22,(10):1449-1453.

VITA I

Chris Tsang is currently in fourth year computer engineering at the University of Manitoba. He was born in 1979 in Hong Kong and immigrated to Winnipeg, Manitoba in 1985. From 2000 -2001 he was an intern for the OC-48 verification department of Nortel Networks in Ottawa. After graduation, he plans on being employed in Winnipeg or starting his studies for a Masters in Engineering. One day, he hopes to be getting a Masters in Business Administration also.

VITA II

Gurinder Kler is currently completing his 4th year of Computer Engineering at the University of Manitoba. He was born in 1978 in Winnipeg, Manitoba, Canada. After receiving his high school diploma from Maples Collegiate in 1996, Gurinder enrolled in the Faculty of Science at the University of Winnipeg. Realizing that was not his thing, in 1997 he enrolled into the Faculty of Engineering. After completing his third year of Engineering, Gurinder opted to go with the internship program at the University of Manitoba. He received a job with Nortel Network's 10Gb/s High Speed Module Development department. Gurinder enjoyed his work, as he got to go design two test boards for the OC192 RF modules. Gurinder would like to continue his engineering career in Winnipeg.