University of Southern Queensland

Faculty of Engineering & Surveying

Stereo Vision for Webcams

A dissertation submitted by

Adam G Cox

in fulfilment of the requirements of

Courses ENG4111 and ENG4112 Research Project

towards the degree of

Bachelor of Engineering - Computer Systems

Submitted: October, 2011

Abstract

This project describes the developmental process of creating and testing a webcam based image capture platform and library for determining the suitability of using webcams for stereo vision. Research and development has taken place in the field of stereo vision for many years, however the costs involved with capture devices and computer hardware has limited the access to the field. This has led to the requirement of a low-cost image capture system that is capable of operating multiple capture devices simultaneously for the application of stereo vision.

The image capture software was developed in the Microsoft Windows operating system environment using the Microsoft DirectShow application programming interface to access and capture video frames. The Camera Calibration Toolbox was used to perform webcam calibration and image rectification to the captured images. Stereo processing was then applied with the Dense Stereo algorithm developed by Abhijit Ogale. The stereo disparity results were analysed for accuracy and precision to determine the suitability of the webcam for stereo vision applications.

The outcome of the development and testing does confirm that webcams can be operated simultaneously and that they can provide a suitable platform for stereo vision. This outcome will increase the accessibility into the research of stereo vision, without the excessive costs that have previously been associated in this field.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Adam G Cox

0050040075

_____

Signature

_____

Date

Acknowledgments

I would like to acknowledge and thank Dr. John Leis for his guidance and support throughout this project. I would also like this opportunity to thank the Faculty of Engineering and Surveying and all the lecturers that have assisted me over the duration of my program.

Adam G Cox

University of Southern Queensland
October 2011

Contents

List of Figures

List of Tables

Glossary of Terms

API	Application Programming Interface
CCD	Charged Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
COM	Component Object Model
CPU	Central Processing Unit
DLL	Dynamic Linked Library
EA	Engineers Australia
FPGA	Field-programmable Gate Array
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HD	High Definition
IDE	Integrated Development Environment
JPEG	Joint Photographic Experts Group
LED	Light Emitting Diode
LIDAR	Light Detection and Ranging
OS	Operating System
OTS	Off The Shelf
PCI	Peripheral Component Interconnect
SDK	Software Development Kit
USB	Universal Serial Bus
UVC	USB Video Class
VFW	Video For Windows
V4L	Video4Linux
VMR	Video Mixing Renderer

Chapter 1

The Utilisation of Webcams for Stereo Vision

Stereo vision research and development has been carried out for almost half a century. It has been the objective of researchers to provide a system that is capable of robust, precise and accurate stereo vision for a large range of applications. While the most common applications for stereo vision have involved robotics and machine vision, new applications have been emerging in remote machine access and personal entertainment.

Accessibility has long been a limitation in the filed of stereo vision research and development, as the cost involved with purchasing professional image capture devices has not been feasible. The requirement has existed for a low-cost image capture platform capable of simultaneous image capture that can be used for computer vision applications including stereo and multiple camera vision processing.

A solution to the accessibility issue will be investigated that includes the implementation of low-cost and off-the-shelf consumer webcams. The consumer webcam has been selected as it has evolved over the years to become a highly capable video and image capture device.

Existing approaches in developing an image capture platform have been limited by the lack of functionality for operating two or more webcams and allowing image cap-

ture simultaneously. The developer has usually had the task of creating the software from scratch, as existing packages are either too expensive or are produced for a niche application and for specific hardware.

By developing an image capture platform it will be possible to determine the suitability of the webcam for high demanding tasks, such as stereo vision. This will involve the investigation and application of calibration, image rectification and disparity mapping processes for the analysis of stereo vision suitability.

Combining webcams with the increasingly powerful desktop personal computer may provide the potential for a low-cost capture platform for research and development applications in the field of stereo vision. The requirement for a robust, accurate, economic and accessible system still exists and this project suggests that a solution is achievable.

1.1 Project Aim

The project aims at providing a solution for the simultaneous operation and image capture from consumer webcams and investigating the suitability of webcams for stereo vision applications. The project will involve the creation of a software solution for an image capture platform that can be used as a stand-alone application or implemented into a library for use by other applications. It is also intended that the capture platform will be used in further research projects.

The suitability of using webcams for stereo vision applications will be researched and tested to identify any limitations that may exists. The research will not only cover stereo processing, but will include calibration and image rectification techniques and processes, that will be included in the testing stages for the project.

1.2 Project Objectives

The overall project has been assessed and broken into key objectives for completion:

- Research stereo and multiple camera vision systems including occlusion reduction and depth extraction techniques.
- Research the feasibility of operating two or more USB webcam devices simultaneously on a single personal computer.
- Design of a software access and control system for multiple webcam image acquisition and edge detection.
- Research calibration and stereo disparity processing techniques and the feasibility of applying them to the image acquisition system.
- Analyse captured image data for edge detection and surface processing accuracy.
- Design and implement image acquisition library into Dynamic-link library format for access by other applications.

Additional objectives will be commenced if time and resources permit:

- Investigate the feasibility of implementing the software system on multiple OS platforms.
- Extend application research into infrared cameras and game based controllers.
- Design of software functions for multiple webcam calibration and stereo disparity processing.

1.3 Overview of the Dissertation Structure

The dissertation is organised as follows:

- **Chapter 1** - Provide an introduction for the project and detail on the aim and objectives.
- **Chapter 2** - The existing literature and approaches will be reviewed to determine the projects feasibility and methodology. Basic concepts related to the project will also be discussed.

- **Chapter 3** - The methodology and approaches for the project will covered and provide an overview of how the project will be completed. Risk assessment and sustainable practices will be detailed in relation to the project.
- **Chapter 4** - The selection of the necessary hardware for both the webcams and computer testing platforms will detailed.
- **Chapter 5** - The overall development stage for the image capture platform and library will be outlined. The development stage will provide information on key areas that are critical for the successful operation of the software.
- **Chapter 6** - The processes involved with testing the image capture platform, library and stereo vision processing will be discussed.
- **Chapter 7** - The chapter will discuss the findings and results from testing stage.
- **Chapter 8** - The research conclusions will be covered and the identification of further research opportunities discussed.

Chapter 2

Literature Review

2.1 Chapter Overview

Stereo vision systems have been research and developed for over 40 years (Narasimha 2010). Early research was limited however to the technology and computing power at the time. This led to much research being carried out with low resolution random dot stereograms (Julesz 1964, p. 357). Current computer systems and capture devices have the potential for high resolution and real-time stereo vision processing for applications requiring 3-dimensional environment information.

This chapter investigates and reviews the existing literature and approaches for developing a computer based stereo vision system. This includes investigating the feasibility of implementing OTS webcams into a stereo vision system.

2.2 Computer Based Stereo Vision

Single image and video capture devices obtain two-dimensional images, thus it is necessary to recover the third dimension from multiple images captured of the same scene from two or more devices (Mubarak 1997, p. 111). Stereo vision is able to achieve the recovery of the third dimension through stereo correspondence and triangulation.

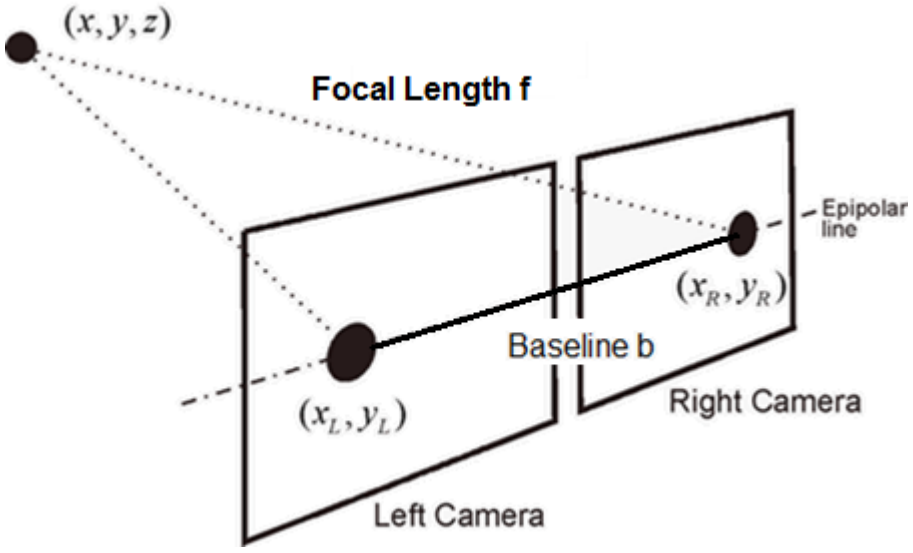

Figure 2.1: Overview of a stereo vision system

The basic stereo configuration is detailed in Figure 2.1, which depicts the relationship of the two camera system with the object of interest.

Where:

- Image Matrix = xL, xR
- Disparity (D) = $xLxR = bf/Z$
- Depth (Z) = $bf/(xLxR)$

Numerous stereo algorithms have been developed, evaluated and documented over the years (Scharstein & Szeliski 2001) that continuously aim at retrieving accurate and reliable 3-Dimensional data. Common stereo algorithms including Banard's Stereo (Mubarak 1997, p. 117), Marr-Poggio cooperative stereo (Forsyth & Ponce 2002, p. 330) and the Horn and Ikeuchi algorithms (Forsyth & Ponce 2002) are widely documented in computer vision literature.

Each algorithm performs differently depending on the image data, for example it was found (Forsyth & Ponce 2002) that the Marr-Poggio approach works well on random dot stereograms, but not on natural images. A gross simplification is that these algorithms complete disparity calculations in a similar method by finding correlating points

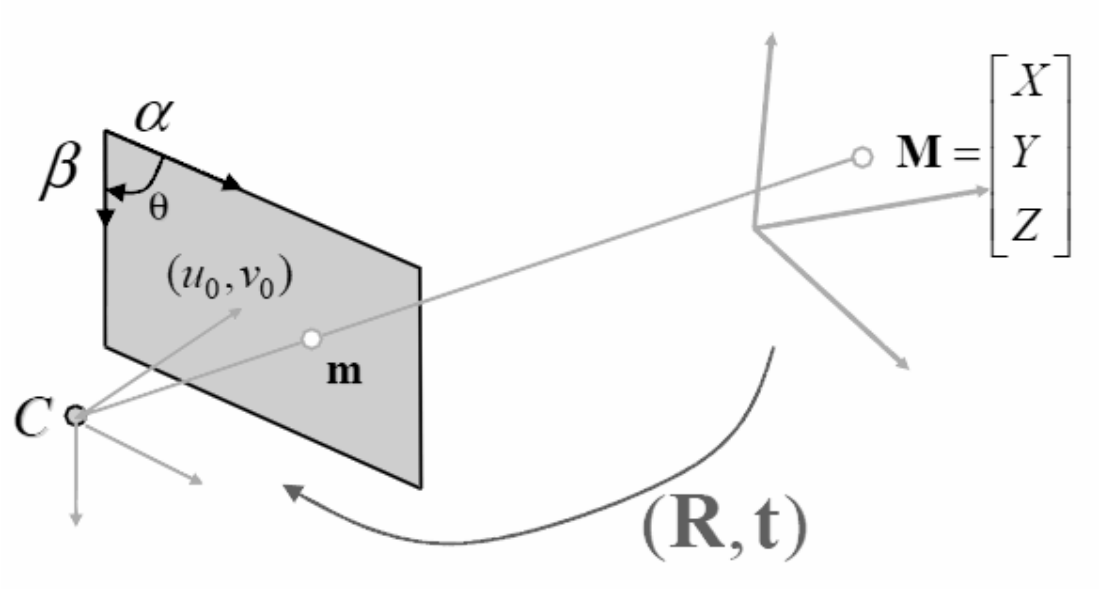

Figure 2.2: Pinhole camera model

of uniqueness between the left and right images.

2.2.1 Camera Calibration and Image Rectification

Camera calibration is an important process for the retrieval of accurate 3-Dimensional data from images (Davies 2004). The function of camera calibration in a stereo vision system is to retrieve the intrinsic and extrinsic parameters of the each individual camera and then the combined camera system. Intrinsic parameters are used to model the imaging process, and extrinsic parameters are used to model the cameras location in its environment (House & Nickels 2006).

Zhang (2000) describes the common conventions used for the describing the calibration approach by using the pinhole camera model in Figure 2.2, which describes the relationship between the image plane and the coordinates of a 3-Dimensional point where:

- Denoting a 2D point as a vector $m = [u, v]^T$.
- Denoting a 3D point as a vector $M = [X, Y, Z]^T$.

- The augmented vector is defined by $\tilde{x}$ and an additional `1` is added to the last vector element $m = [u, v, 1]^T$ and $M = [X, Y, Z, 1]^T$.
- When an image is taken, the 3D point `M`, denoted by `m` is formed by an optical ray from `M` intersecting the image plane and passing through the optical center of the camera `C`.
- The three points `M`, `m` and `C` are collinear.

The relationship between the real world 3D point `M` and the captured projection point `m` is determined by:

$$s\tilde{m} = A[R, t]\tilde{M} = P\tilde{M} \tag{2.1}$$

$$A = \begin{vmatrix} \alpha & \gamma & u_0 \\ 0 & \beta - e & v_0 \\ 0 & 0 & 1 \end{vmatrix}.$$

$$P = A[R, t] \tag{2.2}$$

Where:

- `s` is an arbitrary scale factor.
- `[R, t]` is the extrinsic parameters, which represent the rotation and translation relating the world coordinate system to the camera coordinate system.
- `A` is called the camera intrinsic matrix with `(u0, v0)` the coordinates of the principal point.
- α and β are the scale factors with regards to the image `u` and `v` axes,
- γ represents the parameter describing the skew of the two image axes.

The resulting 3 X 4 matrix `P` is defined as the camera projection matrix. This matrix combines both the intrinsic and extrinsic properties that can be applied to rectification of the captured image pairs.

Medioni and Kang (2004) describes a popular technique that consist of using a checkerboard pattern for calibration. The steps involved are:

- Detecting the corners of the checker pattern in each captured image.
- Estimating the Camera projection matrix `P`.
- Recovering the intrinsic and extrinsic parameters `A`, `R` and `t` from the projection matrix `P`.

Camera calibration is necessary before the image pairs can be rectified, as the calibration process determines the image transformational information (R. Guerchouche 2008). The rectification process provides corrections for radial lens distortion, the principal point and focal lengths. Guerchouche (2008) reports that rectification errors are usually a result of poor placement of the calibrating image pattern, which leads to blurred images and reduces edge or corner detection accuracy.

2.2.2 Occlusion Reduction

Stereo vision systems are usually developed with two cameras that create a stereo pair of images. This has limitations due to occlusion where pixels from one image do not have a match in the correlating stereo pair image (Zitnick & Kanade 1999, p. 675). This can be caused by foreground objects blocking background objects in a scene. It has been documented (Zitnick & Kanade 1999) that a typical approach to deal with occlusion is bidirectional matching that uses disparity mapping interpolation to improve accuracy.

Research has also been carried out into other occlusion reducing methods. It has been reported (Chen & Davis 2000) that limited success was achieved through varying the camera placement. This has resulted in a trade-off between resolution and robustness

of the system. Using trinocular vision, where three cameras are used has been shown to achieve good results. Asensio and Montano report that this method has the advantage of improved accuracy with stereo matching; however it requires greater resources to achieve.

2.3 Current Computer Based Stereo Vision Systems

Current stereo vision systems tend to be designed around a specific niche application with a strong emphasis on machine vision. This results in a system that has software and hardware dependencies unique to the niche application. Attempts to create user friendly and accessible GUI's or software packages still tend to be bounded to a single OS platform being either Windows or Linux with no cross compatibility (Castejon 2009).

Other alternatives including the OpenCV library only provide C and C++ libraries for stereo vision systems (Bradski & Kaehler 2008). MATLAB has also been used as a software interfacing package; however this results in a dependency to MATLAB and does not provide a stand-alone user interface.

It has been identified that the necessity for a system to be low-cost is usually overshadowed by the requirement of accuracy and robustness. It has been reported (Murphy, Lindquist, Rynning, Cecil, Leavitt & Chang 2007) that this has resulted in a cost prohibitive technology and limits access for future research and development. The USB webcam can provide a platform that can capture Full HD resolution video at 30 frames per second. The USB webcam also provides this functionality at a cost $10 - $150 per unit, which is an attractive option when compared to existing systems such as LIDAR that can cost $7,000 - $20,000 (Murphy et al. 2007, p. 333). FPGA devices provide another alternative, but low cost configurations require development, interfacing and housing.

2.4 Operation of Multiple Webcams

It is necessary to operate two or more webcams simultaneously and to capture video frame data for further processing. In order to achieve this objective it will be necessary to determine the feasibility of webcam operation in relation to both the hardware and software components.

2.4.1 USB Interface Capabilities

The use of two or more webcam devices places increased strain on transfer methods between the webcams and computer system. All current webcams and computer systems provide USB connectivity of version 2.0 to meet the bandwidth requirement of streaming video data.

USB 2.0 provides suitable transfer speeds of 480 Mbps (Axelson 2009, p. 12), with the release of USB 3.0 it is likely that hardware manufactures will start adopting the newer standard in both webcams and computer systems that can provide transfers of up to 5 Gbps (Axelson 2009, p. 13).

2.4.2 Application Programming Interface

The interface between the webcam device drivers and the OS requires compatibility with new and old webcam devices while also being able to stream from two or more webcams. This project will involve the development of an image capture platform capable of running on Microsoft Windows OS's, however non-Windows OS's will be researched for the potential of future work.

2.4.3 Microsoft Windows API's

Microsoft has developed many multimedia APIs to operate on their Windows based OSs. The first notable API was Video For Windows (VFW), which was provided with the release of Windows 3.1 and the later Windows 95 OS. VFW provides limited

format and device support (Microsoft 2011*a*) due to it's early development and release. Subsequently VFW has been superseded by the DirectShow API (Microsoft 2011*b*).

The Directshow API was released in 1996 under the name ActiveMovie, however it was renamed to DirectShow in 1998 to distinguish it's link with the DirectX API. DirectShow has had many revisions since release with the notable inclusion of the Video Mixing Renderer (VMR) filter (Microsoft 2011*a*). The VMR allows the mixing of multiple video sources into a single video stream.

DirectShow provides flexibility and high quality capture when working with multimedia streams (Microsoft 2011*a*). DirectShow applies filters for the control of capture devices. Multiple filters are able to be accessed simultaneously and are managed by the DirectShow filtergraph. This capability will allow multiple webcams to operate simultaneously.

DirectShow is still supported in the latest Windows OS - Windows 7, however it will be superseded by the Microsoft Media Foundation API. Media Foundation was released in 2007 with the Windows Vista OS and is currently supported alongside DirectShow (Microsoft 2011*c*). Media Foundation will include support for emerging high definition devices and formats.

The support and documentation for both VFW and Media Foundation is limited due VFW being superseded and the relatively short period that Media Foundation has been in use. DirectShow provides the compatibility, functionality and support that the alternative API's lack.

2.4.4 Linux and Apple API's

Non Microsoft OS's have a substantially smaller market share with Microsoft controlling over 90% of the market (Net Market Share 2011). Both Linux and Apple Mac OS distributions also include some form of multimedia API for video capture.

Linux OS distributions can access and manipulate multimedia streams with the Video4Linux (V4L) API. The V4L API provides similar capability to DirectShow by allowing the op-

eration of multiple devices and supporting common image and video formats (LinuxTV 2009).

A limitation of V4L is the small number of supported hardware devices, as some manufactures do not provide device drivers for Linux. To provide improved device support Linux provides a default USB Video Class (UVC) driver (Ubuntu 2011).

Apple provides the QuickTime API for both Windows and Mac OS. The API provides over 2500 multimedia functions for manipulating and controlling video and audio data (Appple 2011). The Quicktime API supports numerous languages including C, C++, C#, Java and supports Component Object Model (COM) and .Net frameworks. Quicktime allows multiple media sources to operate simultaneously and to allow image capture.

2.5 Infrared Based Game Controllers for Stereo Vision Systems

The method for controlling actions and movement inside a video game environment has changed in recent years with the release of the Nintendo Wii gaming console in 2006 and with the Xbox Kinect gaming controller in 2010. The controllers provide a wireless and infrared method of object tracking. Both the Wii and Xbox controllers provide a feasible alternatives for stereo vision systems.

2.5.1 Nintendo Wii

The Wii controller contains conventional buttons along with an infrared camera. A sensor bar containing two clusters of infrared LEDs is positioned near the output display. The controller works by detecting the two cluster points and calculates the relative position and direction of movement of the controller with respect to the display (Hay, Newman & Harle 2008).

The controllers optical sensor consists of a CCD capable of a resolution of 1024x768

combined with an infrared filter capable of sensing up to four infrared points at a speed of 100 Hz (Cuypers n.d.).

(Lee 2008) reports that it is feasible to create an inexpensive stereo vision system with Wii controllers, while using more then two controllers may reduce occlusion issues. Existing research has also concluded that a level of precision under 3 mm is possible with a calibrated Wii-based system (Cuypers n.d.).

2.5.2 Xbox Kinect

The Kinect controller is gaining popularity in the development community, with the official support and release of a software development kit by Microsoft.

The operation of the kinect controller involves an infrared projector and camera incorporated into the same piece of hardware. The infrared projector projects a known, near-random pattern of infrared dots onto its field of view. The dots are captured by the infrared camera that is offset by 25 mm from the projector on its epipolar axis. The depth of an object is determined by the level of dot disparity between the known projection pattern and the dot pattern captured by the camera (Ball & Taschuk 2011).

The Kinect controller provides advantages over existing stereo vision systems, as it can handle ambient lighting condition as it does not register light in the visible spectrum (Carmody 2010) and (Ball & Taschuk 2011) found that it is feasible to develop a vision system with a single capture device capable of depth estimation.

2.6 Chapter Summary

From the conducted research it has been determined that the requirement does exist for an accessible and low cost image capture system implemented from OTS hardware, capable of collecting image data for stereo processing.

There has been a wide range of research into the field of stereo vision; however a large proportion of research has focused on machine vision or other niche applications. Cam-

era calibration and image rectification has been researched thoroughly an its importance appreciated for the successful implementation of a stereo vision system capable of depth estimation.

The use of infrared gaming controllers for an economical and alternative stereo vision system is feasible and development is relatively new and progressing rapidly.

From the determined feasibility this project aims at implementing multiple USB web-cams via an API into an image capture platform. The capture platform will provide the capability of simultaneous image capture that will be used for further stereo processing and the extraction of environmental depth information.

Chapter 3

Methodology for Software Development and Stereo Vision Evaluation

3.1 Chapter Overview

This chapter covers the methodology and approach taken for the development of the multiple camera platform and stereo vision evaluation. This chapter also provides information on the risks and consequential effects involved with the research and development.

3.1.1 Project Methodology

To determine the methodology of the project it was necessary to develop an understanding of how the objectives would be achieved and therefore a logical breakdown of the main tasks.

The tasks involve creating an image capture platform and library for webcams that can be used to capture images from streaming video. The captured images can be used for

further stereo processing for depth estimation. For this to be achieved it is important that the correct interactions between the systems software and hardware layers are implemented.

The system will require at least two webcams for stereo vision processing. The webcams will require a physical mounting, so that they share a common scene of interest during the image capture process. The webcams need to be interfaced with the desktop system, so that the video streams can be accessed and image data retrieved from video frames.

A software platform with an intuitive GUI will need to be developed to allow the user to operate the webcams and capture image data for later processing. The GUI will need to provide the user with enough control to add or remove webcams and to adjust the webcam properties.

Calibration of the webcams will be required to correct any variation in physical orientation and location. This process will have to retrieve the cameras intrinsic and extrinsic parameters for accurate stereo processing.

A software process will need to be applied that is able to perform image rectification and edge detection. This is necessary to align the image pair planes and to identify points of correlation between the images, which will result in a higher level of stereo vision processing accuracy.

The next software process will apply the stereo processing that will result in disparity mapping. The generated disparity maps will be evaluated for accuracy and suitability to real world applications.

The development of a webcam access software library will be carried out for testing in other programming applications or existing software packages. The outlined methodology requires that all the necessary tools, such as IDEs have been configured and that all necessary resources obtained.

3.2 Task Breakdown

The project methodology can be refined into eight major tasks:

1. Stereo webcam mounting
2. Webcam Interfacing in both hardware and software
3. Development of the multiple camera access platform
4. Software library development for webcam access
5. Webcam image capture
6. Webcam calibration to correct intrinsic and extrinsic parameters
7. Image pair rectification and edge detection
8. Stereo processing and disparity mapping

These tasks are identified as being key milestones for development and as and indication of the projects progression.

3.3 System Operation

The overall system in Figure 3.1 consists of two key components:

1. Webcam access and capture through the GUI platform or software library
2. Stereo processing of captured image pair data.

The operation of the system is reflected in the developmental task breakdown and how the software components interact.

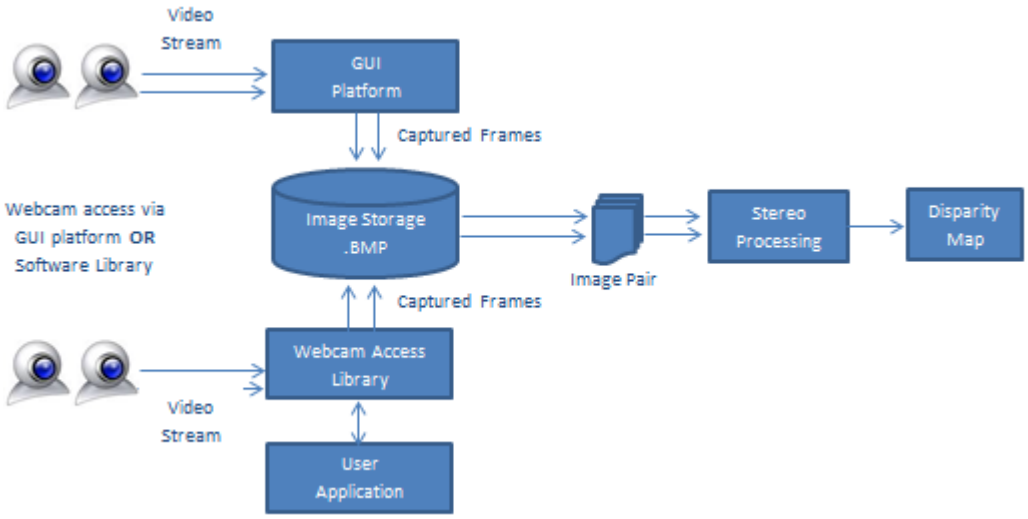

Figure 3.1: Overview of the system operation

3.4 Task Analysis

The methodology for the eight major tasks defined for this project will be analysed to determine the approach to be taken and the resources required.

3.4.1 Stereo Webcam Mounting

The webcams will require mounting during the image capture and stereo processing stages of the project. The selected mount must provide a solid foundation for the webcams to reduce any unwanted webcam movement or vibrations. The mount must also accept different webcam models and configurations. The mount will be situated in an office environment and will also be limited to the physical space available.

Resources Required:

- Physical mounting for webcams.

3.4.2 Webcam Interfacing in Both Hardware and Software

The hardware interfacing for the webcam should be straight forward as completed research has found that the USB interface is common amongst all webcam and motherboard manufactures. The software interfacing will be more challenging, as code needs to be developed that will allow the webcam to interact with the OS, image capture platform and library. The software interface will have to include a suitable API that allows multiple webcams to operate simultaneously on a single desktop computer system.

Resources Required:

- Webcams with USB connectivity.
- Desktop computer system(s) for testing that include USB connectivity.
- API for software interfacing.

3.4.3 Development of the Multiple Camera Access Platform

The image capture platform will require a functional and easy to use GUI for user interaction. This will involve writing a Windows application that provides functionality for webcam selection, playback, edge detection and image capture. The user should be able to control the operation of all the functionality components, which would provide a better user experience. The platform should allow access of at least three webcams for image capture and streaming. This will be a capability that has not been found in other webcam applications that limit the number of webcams and image capture to only one or two devices.

Resources Required:

- Webcams
- Selection of programming language.
- IDE for software development.

3.4.4 Webcam Image Capture

This will also involve the creation of software that is capable of capturing the video frames from the streaming webcam video. The captured images should be able to be stored in a common image format that is supported by most applications. The capture will need to be simultaneous with little time separating each webcam capture.

Resources Required:

- Webcams
- Selection of programming language.
- IDE for software development.

3.4.5 Webcam Calibration for the Correction of Intrinsic and Extrinsic Parameters

A calibration process will need to be acquired that can be applied to the captured images from the image capture platform or library. The calibration must be able to determine the intrinsic and extrinsic parameters of the webcams and environment. The calibration data must be able to be collected and stored for further processing including image rectification.

Resources Required:

- Existing calibration software package.

3.4.6 Image Pair Rectification and Edge Detection

The image rectification process must operate similarly to the calibration process, where the process can be applied to the captured images from the platform. This will involve writing a software process, modifying provided code by Dr John Leis or using an available software package.

Resources Required:

- Existing software package capable of handling imported images and calibration data.

3.4.7 Stereo Processing and Disparity Mapping

The suitability of using webcams for image capture and stereo vision is a major objective in the project. The approach for applying stereo processing to the captured images must be robust and accurate. Since numerous stereo algorithms exist it will be necessary to research and select one capable of handling the image format and selected scene type. It may be necessary to save captured images in a format that can be accessed by the stereo vision processing package.

Resources Required:

- Existing Stereo Vision Processing software package.
- Suitable scene for testing.

3.4.8 Software Library Development for Webcam Access

The image capture library development will require a programming approach that includes the required DLL commands. The DLL format will need researching before development can begin. The Library should be able to be tested in both a third party application such as MATLAB; and through a developed program. The library will provide similar functionality to the image capture platform.

Resources Required:

- Selection of programming language.
- IDE for software development.

3.5 Consequential Effects

The possible consequences of introducing a new system into the public have been investigated to ensure the impact is sustainable, safe and ethical (USQ 2011).

3.5.1 Sustainability

EA has provided guidelines for evaluating the sustainability on the environmental, social and economical systems over its full life cycle (Engineers Australia 2011*b*). The proposed stereo vision system will have a minimal impact during its lifecycle. The manufacture of the device and the energy consumption does contribute to the usage of energy and materials developed through non-renewable processes. These issues involving the manufacturers processes are outside of the scope of this project.

3.5.2 Safety

The system will consist of OTS webcams and desktop computer systems that must meet the required Australian Safety Standards.

3.5.3 Ethical Dimensions

EA provides a code of ethics to be related to engineering practices (Engineers Australia 2011*a*). The code of ethics covers integrity, competence, leadership and sustainability. In undertaking this project it is important to incorporate the code of ethics in all areas, to ensure that the final outcome of the project will reflect the best interests of the community.

3.6 Risk Assessment

The risks and possible hazards associated with this project are similar to those involved in any IT environment. It is important to identify and reduce the likelihood of risk associated with the project and its use by both the operator and future users.

3.6.1 Risk During the Execution of the Project

The following hazards have been identified as those possibly encountered during the execution of the project:

1. Occupational Overuse Syndrome:
    - Evaluation - These types of injuries occur after prolonged use of computer peripherals including the mouse and keyboard. Injuries include tendonitis and carpel tunnel syndrome.
    - Control Take regular breaks from prolonged usage. Make sure breaks include standing and walking for several minutes.
2. Lighting:
    - Evaluation - Poor lighting including glare can cause eye strain and headaches for the operator.
    - Control Correctly position monitors so they are adjacent to windows. Close blinds or curtains if necessary and possibly use a screen filter.
3. Stress:
    - Evaluation - Long hours and insufficient breaks can cause stress to the operator and therefore increase the likelihood of other conditions.
    - Control Healthy diet, exercise and break up large tasks into smaller and more manageable tasks and take regular break.
4. Radiation:

- Evaluation - Radiation from electrical devices such as wireless routers produce low levels of radiation and operate without risk unless the device has become damaged.
- Control Do not used damaged electrical devices, place any wireless emitters away from users or disable when not in use.

5. Noise:

   - Evaluation - Office equipment produces low levels of noise that is harmful to human hearing; however damaged equipment may increase the level of noise and should be investigated.
   - Control Correct placement of noise producing devices, so there is a physical distance between the user and device.

6. Electrocution:

   - Evaluation - All mains power equipment in commercial use is to be tested and tagged to reduce the use of damage equipment. It is possible that damage equipment can be used, which
   - can cause serious risk to the user. Control Avoid using food or drinks around electrical equipment, never dismantle equipment unless qualified and do not place power chords in areas of high traffic.

7. Heavy Lifting:

   - Evaluation Some computer systems and monitors can be heavy and cause injuries.
   - Control Follow correct lifting instructions provided by manufactures, always lift with a straight back and if necessary get assistance from others.

3.6.2 Risk Beyond the Completion of the Project

The same hazards found during the execution of the project apply to the future users of the stereo vision system.

Level	Code	Description
1	Rare	Event may occur only in exceptional circumstances
2	Unlikely	The event may occur at some time, once in 10 years
3	Moderate	The event should occur at some time, once in 3 years
4	Likely	The event will probably occur in most circumstances, once a year
5	Almost Certain	The event is expected to occur in most circumstances, many times a month

Table 3.1: Hazard Likelihood

Level	Code	Description
1	Insignificant	First Aid and/or Minor Equipment Damage
2	Minor	First Aid and/or Major Destruction of Equipment
3	Moderate	Moderate Injury or Illness requiring examination
4	Major	Major illness or temporary disability requiring hospitalisation
5	Catastrophic	Death or permanent disability

Table 3.2: Hazard Consequence

Consequence	5, Almost Certain	4, Likely	3, Moderate	2, Unlikely	1, Rare
5, Catastrophic	25	24	23	22	18
4, Major	21	20	17	16	11
3, Moderate	19	15	14	10	7
2, Minor	13	12	9	6	5
1, Insignificant	8	4	3	2	1

Table 3.3: Risk Matrix

Hazard	Likelihood	Consequence	Rating
Occupational Overuse Syndrome	3	3	14
Lighting	4	1	4
Stress	4	2	12
Radiation	1	2	6
Noise	4	1	4
Electrocution	2	5	22
Heavy Lifting	4	3	15

Table 3.4: Risk Analysis

3.6.3 Risk Summary

Overall the project will involve a low to medium level of risk. It is important to be aware of the risks and take steps to reduce any consequences associated with them.

3.7 Research Timeline

The project timeline outlining the managment of tasks and achievement of milestones is located in Appendix B.

3.8 Chapter Summary

The methodology and approaches for the major project tasks have been discussed in this chapter. The task analysis outcomes will be implemented throughout the projects progression.

The size and background of this project results in a low risk and sustainable outcome. There are a small amount of hardware and software resources required, however most have already been obtained or are freely available.

Chapter 4

Test Platform and Webcam Selection

4.1 Chapter Overview

This chapter provides information on the hardware selection criteria for the desktop test platform and webcam selection. The selection of webcams and the test platform is important, as it is a project requirement to utilise low-cost and OTS hardware for the test PC and webcams.

4.2 Test platform Requirements

It is an important requirement that the image capture platform and stereo processing can be operated on a commonly found desktop PC. The term common referring to a desktop system of approximately five years of age. This is necessary to allow a greater level of accessibility for potential users.

4.2.1 USB Connectivity and Interfacing Requirements

The test platforms will require at least two available USB 2.0 ports for webcam connection. Two ports will be required for testing the stereo abilities of the captured image pairs, while up to five ports will be required when testing the webcam enumeration and detection process.

The USB 2.0 standard was released in 2000 and provides transfer speeds of 480 Mbit/s, which is necessary for simultaneous video streams and image capture.

4.2.2 Processor Configuration

The video streaming, image capture and stereo processing will all be handled by the system CPU. There is no utilisation of multi core processors during testing, which will reflect the capabilities of some older desktop computers.

CPU's manufactured by both Intel and AMD will be selected, as each processor provides a different architectural design and features.

4.2.3 Operating System

Testing will be based on the Microsoft Windows OS. This will also increase the accessibility of the software. The image capture platform has been designed to operate on the Microsoft Windows XP, and Windows 7 OS's in both 32 and 64 bit configurations.

Other OS exists from alternative manufactures, however Microsoft has the largest OS market share, which results in a high level of accessability.

4.3 Webcam Requirements

The most important requirement in selecting webcams for this project is the unit price. The objective exists that the end user be able to use low-cost and commonly available

consumer webcams for operation.

Considerations of less importance include the webcam mounting and USB cable length.

4.3.1 Webcam Costs

Commercially available webcam products are priced between $10 - $150 depending on certain features and manufacture. It is therefore necessary that the user can install an operate webcams that have been selected from the low end of the price spectrum.

4.3.2 Webcam Compatibility

There are two important areas to investigate for webcam compatibility:

- The software layer
- The hardware layer

On the software layer, Webcams interface with the OS through device specific drivers and an API. Ensuring that a high level of compatibility exists has been covered by using the Microsoft Windows OS and the Microsoft Directshow API, which is discussed in Chapter 5.

The physical connection on almost all webcams is via the USB 2.0 interface. This provides a capable interfacing technique as discussed in Section 4.2.1.

4.3.3 Webcam Build Quality and Properties

Webcam construction and build quality varies amongst manufacturers, webcam models and even between two identical webcams. The lower cost webcams commonly utilise a CMOS based image sensor and plastic optics, where as the more expensive webcams utilise a CCD sensor and higher quality optics.

These differences in construction will be analysed and evaluated for image quality and the affects had on stereo processing.

4.4 Selected Test Platforms

Three test platforms were selected for the testing of the image capture platform and software library, while a single system was selected for the stereo processing. The systems were selected based on there high level of accessibility and configuration. Test Platform 2 will be used for testing the stereo processing, as it has the lowest system performance out of the three platforms.

The system specifications for each system are detailed in Table 4.1, Table 4.2 and Table 4.3.

Test Platform: 1	
Operating System	Windows 7, 64 bit
CPU Model	Intel Q6600
CPU Speed	Quad Core @ 2.4 GHz
Memory	4 Gb
USB Standard	2.0

Table 4.1: Test Platform 1: Properties

Test Platform: 2	
Operating System	Windows 7, 32 bit
CPU Model	AMD 9650
CPU Speed	Quad Core @ 2.3 GHz
Memory	2 Gb
USB Standard	2.0

Table 4.2: Test Platform 2: Properties

This selection of test platforms ensures a good representation of common desktop computer system configurations.

Test Platform: 3	
Operating System	Windows XP, 32 bit
CPU Model	Intel Q6600
CPU Speed	Quad Core @ 2.4 GHz
Memory	4 Gb
USB Standard	2.0

Table 4.3: Test Platform 3: Properties

4.5 Selected Webcams

Two pairs of webcams were selected for operation and testing. These include the Logitech C200 and Logitech C905. The webcams provide a good comparison of differently priced and constructed webcams that are currently available on the market.

The webcam specifications are listed in Table 4.4 and Table 4.5.

C200 Webcam	
Manufacturer	Logitech
Sensor	CMOS
Max Video Resolution	640 x 480 Pixels
Frames Per Seconds	30
Connection Interface	USB 2.0
Price	$10.00

Table 4.4: C200 Webcam Properties

The manufacturer Logitech was selected due to their large market share and availability.

C905 Webcam	
Manufacturer	Logitech
Sensor	CCD
Max Video Resolution	1600 x 1200 Pixels
Frames Per Seconds	30
Connection Interface	USB 2.0
Price	$55.00

Table 4.5: C905 Webcam Properties

4.6 Chapter Summary

The hardware requirements and selection were discussed for the project. Suitable desktop computer test platforms were selected to provide a good range of hardware configurations during testing.

The requirements and selection of the webcams was completed, which consists of two webcam pairs based on the Logitech C905 and C200 webcams.

Chapter 5

Image Capture Platform and Library Development

5.1 Chapter Overview

This chapter discusses the developmental process involved with the project. Development involved two main tasks; the image capture platform GUI and the image capture library. Each task required different knowledge and resources for successful execution.

The software requirements and selection for development are also discussed to provide details on how they impacted on development process.

5.2 Software Requirements and Selection

The development tasks involved with the project focus on providing a software solution for webcam access and image capture. This required the selection of:

- A suitable programming language
- An Integrated Development Environment

- An Application Programming Interface

Each software requirement had to be solved based on, existing familiarity, functionality, accessibility and time required for competent operation.

5.2.1 Programming Language

Both the image capture platform and library had to be programmed in a language that could be compiled and operated on the Microsoft Windows OS. It also had to be suitable for developing the DLL format library.

Other considerations for the selection of the programming language included speed of execution and cross-platform portability. This project will not involve the development of a software solution for non Windows based OS's, however it maybe a task in future projects. The potential for future project work was therefore an important consideration when selecting a programming language.

The most suitable language that fulfilled the requirements was C. Other languages would also be able to achieve the same outcome, however the C language has been in use for almost 40 years and is well documented, supported, efficient and portable.

5.2.2 Integrated Development Environment

For a more efficient development approach it was necessary to select an IDE capable of editing and compiling C source code.

After investigation and recommendation, the Wedit IDE was selected. Wedit is a simple editor interface capable of syntax highlighting, which includes the lcc-win32 Compiler system for Windows by Jacob Navia

The included library is capable of developing basic Windows and console applications. This produced some limitations, as additional libraries had to be accessed and linked during development.

5.2.3 Application Programming Interface

The API is required to link the webcam device drivers and the OS together, so webcam access can be achieved. The API must provide the abilities of operating multiple webcams simultaneously and allow for image capture from the multiple video stream.

The Microsoft DirectShow API was selected for this task for the following reasons:

- DirectShow provides improved compatibility and support over VFW and Media Foundation for the selected test OS's.
- The DirectShow architecture provides full manipulation and handling of multimedia tasks.
- Source code examples developed by Dr John Leis have been provided that use the DirectShow API to access a single webcam and provide edge detection processing.

Specific requirements for the DirectShow API are a Windows XP - Service Pack 1 or Windows 7 OS. These OS's include the Microsoft DirectX 9.0 or later API, that incorporates the DirectShow API libraries.

5.2.4 Microsoft DirectShow Operation

DirectShow is able to handle multimedia by using two types of object classes:

- Filters which are the atomic entities and control the media devices attached to the system.
- Filtergraphs control multiple filters connected together.

To access media from a device, the filter uses pins that can either receive an input stream or send an output stream. The number of pins is determined by the device. Figure 5.1 provides an overview of the filter and pins applied to the C905 webcam.

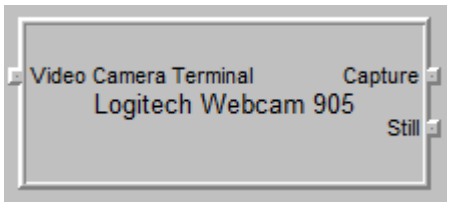

Figure 5.1: Pins attached to filter of the C905 webcam

DirectShow is also based on the Component Object Model (COM), which allows developers to create binary based code components that are re-usable, can be created in multiple languages and accessed by different applications.

COM components are created using Globally Unique Identifiers (GUID) or Universally Unique Identifiers (UUID), which provide an identification method for the software components. An an example of this is:

```
CoCreateInstance(&CLSID_FilterGraph, //Class ID for COM object
NULL,
CLSCTX_INPROC_SERVER,
&IID_IGraphBuilder, // Interface ID
(void **)&pGraphBuilder1); // Filter Pointer

pGraphBuilder1->AddFilter(pSrc1, L"Video Capture");
```

The code example creates a filtergraph `pGraphBuilder1` and attaches the filter `pSrc1`. The filtergraph is encapsulated in the COM `CLSID_FilterGraph`.

The capabilities of the DirectShow API can be broken into three areas, which are based on the types of filter operation:

1. Capture of both audio and video from a live camera device, in this project webcams. This also includes the ability to open a file and treat it as if it were a live multimedia source.

2. When a multimedia source has been selected and the video or audio stream captured, DirectShow filters can transform the media. This includes colour conver-

sion or splitting the media and sending it to multiple filters for further processing.

3. The third capability is rendering of the media to an output device. This includes speakers, monitor display, writing to a disk or outputting to another device.

The three filter types and their relationship with the filtergraph and devices is shown in Figure 5.2.

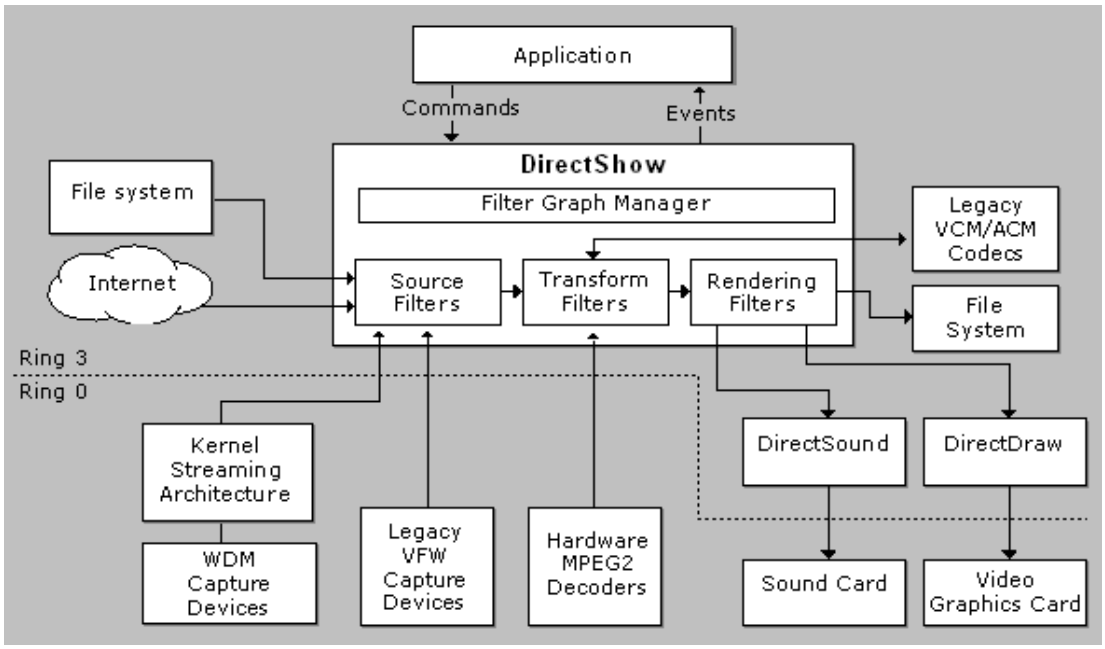

Figure 5.2: Overview of the Microsoft DirectShow operation

5.3 Image Capture Platform Requirements

The image capture platform will provide the user with a stand-alone GUI package that is easy to operate and provides a high level of functionality. The key requirements of the platform include:

- Design of an easy to use GUI with webcam controls.
- List all the available webcams on the desktop system.
- Control the selected webcams properties.

- Provide the user with the option of selecting the webcams they require.
- Display a live video stream from the selected webcams.
- Apply real-time edge detection to the webcam video if selected by the user.
- Capture video frames from the webcam video streams and store them to disk in *.BMP format.
- Upon closing the image capture platform application close all DirectShow objects.

Each of the listed requirements must be addressed during the development stage of the project to ensure that a suitable solution is achieved.

5.3.1 Review of Provided Code Example

An existing code example was provided by Dr John Leis of the University of Southern Queensland. The code allowed the user to access a single webcam and provided playback controls and real-time edge detection to the video stream.

The code provided a working method for accessing a webcam through the DirectShow API on a Windows based OS. Components of the code have been modified and included into the development of the image capture platform.

5.3.2 GUI Layout

It is an important requirement that the user be able to easily work with and operate the platform. This requires a user-friendly and intuitive GUI design that provides quick access to user operated controls.

The overall layout for the platform is comprised of six video windows in a 2x3 configuration. Each window is 320 pixels wide and 240 pixels high. These dimensions were used to conserve the amount of desktop space required during operation. The top row contains three windows for real-time video streams from the selected webcams.

The bottom row displays the captured webcam video with the edge detection process applied. Figure 5.3 shows the final image capture platform GUI.

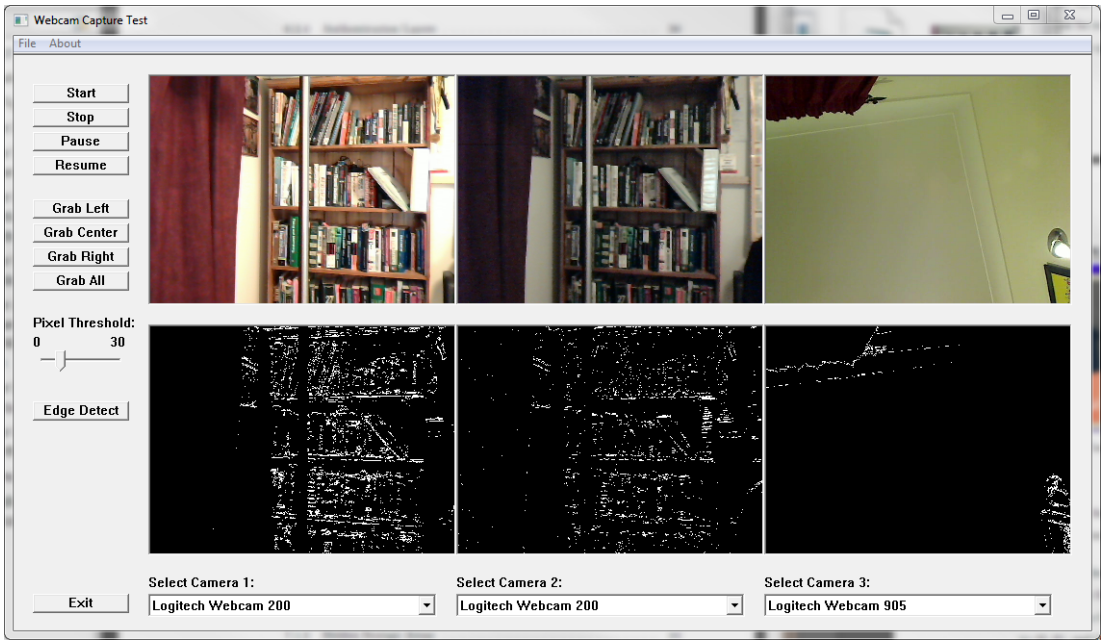

Figure 5.3: Final image capture platform GUI

The GUI is designed to operate up to three webcams simultaneously. It is possible to operate more devices with the DirectShow API, however this limitation was decided upon to conserve desktop space. Each video window is provided with a drop down menu for the individual webcam selection from up to five available webcams.

The GUI provides the user with a selection of controls for the webcam playback, image capture and edge detection function. These controls were implemented as Windows buttons in the left hand side of the GUI by using the `CreateWindowEx` command. An example of this is the stop button:

```
CreateWindowEx(BS_PUSHBUTTON,
                    "button",      // window class name
                    "Stop",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 55, 100, 20,
                    hWnd, (HMENU)IDB_STOPBUTTON,
                    hInstance, (LPVOID)NULL);
```

The playback controls included Start, Stop, Pause and Resume. Playback buttons were linked to the DirectShow `pMediaControl1` command when activated by the user. This resulted in:

- `pMediaControl1->Run()` for Start and Resume controls.
- `pMediaControl1->Pause()` for the Pause control
- `pMediaControl1->Stop()` for the Stop control. DirectShow objects are also released when the Stop control is selected

Image capture button controls were implemented into the GUI for each individual webcam or a single control can be used to grab a simultaneous frame from all running webcams. The operation behind the controls is detailed in Image Acquisition section of this chapter.

The provided edge detection code can be operated by selecting the Edge Detection button control option on the GUI. This enables or disables the edge detection functionality of the platform. The operation behind the edge detection control is detailed in Real-Time Edge Detection section of this chapter.

5.3.3 Enumeration and Selection of Available Webcams

It was necessary to find and enumerate all the available webcams attached to the system when the platform was started. This allows the user to easily select the required webcams for streaming or capture.

The `FindCaptureDevice()` function was developed for the process of enumerating all available webcams. The enumeration of devices was achieved by using the DirectShow command; `CreateClassEnumerator`. The `CreateClassEnumerator` command was defined for video devices by using the `CLSID_VideoInputDeviceCategory` command, which returned a collection device monikers. A device moniker is a COM object that contains information about the enumerated device.

The overall enumeration process operates in a loop structure, outputting the device information into monikers and then assigning the moniker to a filter. In the simplified section of code from the platform development below, the device moniker `pMoniker` is binded to the filter object `pEnumSrc1`:

```
while (pClassEnum->Next(1, &pMoniker, NULL) == S_OK)

pMoniker->BindToStorage(0, 0, &IID_IPropertyBag, (void **)&pPropBag);

pMoniker->BindToObject(0,0,&IID_IBaseFilter, (void**)&pEnumSrc1);
```

The filter object is then used for other operations including the rendering of the video to screen and for configuring the webcam properties. These other operations are detailed throughout this chapter.

5.3.4 Control of Webcam Properties

When the user selects a webcam from the drop down menu, they will have the option to modify the webcams properties. These properties include:

- Webcam image format
- Video resolution
- Frames per second for video streaming

The webcam video resolution can be adjusted by the user, however the webcam video windows are restricted to the dimensions, 320 pixels wide and 240 pixels high. This results in a scaled video been shown in the GUI. The captured image size will be the same as the user selected resolution.

The webcam properties are accessed by querying the pins attached to the device filter. This is detailed in the platform code below:

```
// Enumerate pins from Capture filter.
pSrc1->EnumPins(&pEnum1);
pEnum1->Reset();
pEnum1->Next(1, &m_pCamOutPin1, NULL);

// Pin Properties.
hr = m_pCamOutPin1->QueryInterface(&IID_ISpecifyPropertyPages,
(void **)&pSpecPropPage1);

if (SUCCEEDED(hr))
{
// Code for Property Window
}
```

The code operates by enumerating the pins from the filter object `pSrc1` and assigning the pin object to `m_pCamOutPin1`, which is queried by the command:

`IID_ISpecifyPropertyPages`.

The webcam property window is then displayed for user interaction.

The code developed for the configuration of webcam properties is included in the `InitWebCamCapture'x'()` where `'x'` denotes a webcam number from 1-3.

5.3.5 Simultaneous Webcam Access

The requirement to operate multiple webcams simultaneously resulted in the development of three individual DirectShow filter graph managers that manage each of the three webcam filters.

The main webcam access functions that define and manage the filters are:

`InitWebCamCapture'x'()`

while calls are made from within `InitWebCamCapture'x'()` to the:

`InitializeWindowlessVMR'x'()` and `InitVideoWindow'x'()` functions for the control of the video windows.

NOTE: `'x'` denotes a webcam number from 1-3.

The filter graph manager controls the webcam filter objects along with synchronization and event notification. The filter graph manager is created by defining an instance of the filter graph manager class `CLSID_FilterGraph`:

```
CoCreateInstance(&CLSID_FilterGraph,
CLSCTX_INPROC_SERVER,
&IID_IGraphBuilder,
(void **)&pGraphBuilder'x');
```

After the filter graph manager `pGraphBuilder'x'` was created it was necessary to use the Video Mixing Renderer (VMR) to control the output of the video. By default DirectShow outputs the video to a separate `ActiveMovie` window. It was necessary for this project to confine the video to one of the defined video windows on the GUI.

The `InitializeWindowlessVMR'x'()` function is called within `InitWebCamCapture'x'()` to establish the type of video output rendering for the webcam. To use the rendering features of VMR it was necessary to define an instance of the VMR class `CLSID_VideoMixingRenderer`:

```
CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
CLSCTX_INPROC, &IID_IBaseFilter, (void**)&pVmr'x');
```

It was necessary to create a VMR filter object `pVmr` and attach it to the filter graph manager `pGraphBuilder'x'` to allow the video output to render:

```
pGraphBuilder'x'->AddFilter(pVmr'x', L"Video Mixing Renderer");
```

The video output was required to be windowless and fixed to the defined video windows in the GUI, therefore it was necessary to select the `VMRMode_Windowless` rendering mode offered by the VMR class:

```
pVmr'x'->QueryInterface(&IID_IVMRFilterConfig, (void**)&pConfig'x');
pConfig'x'->SetRenderingMode(VMRMode_Windowless);
pVmr'x'->QueryInterface(&IID_IVMRWindowlessControl, (void**)&VMRpVidWin'x');
```

After the VMR class was created and video rendering style was selected, it was necessary to create the `IID_IMediaControl` and `IID_IMediaEventEx`. `IID_IMediaControl` controls video streaming and contains methods for stopping and starting the graph. `IID_IMediaEventEx` provides methods for managing events from the Filter Graph Manager:

```
pGraphBuilder'x'->QueryInterface(&IID_IMediaControl,
(void **)&pMediaControl'x');
pGraphBuilder'x'->QueryInterface(&IID_IMediaEventEx,
    (void **)&pMediaEvent'x');
```

The capture graph is used for video capture filters graphs and provides easier graph implementation. The `CaptureGraphBuilder2` class was created to control and build the capture graph, which is assigned back to the filter graph manager `pGraphBuilder'x'`:

```
CoCreateInstance(&CLSID_CaptureGraphBuilder2,
NULL,
CLSCTX_INPROC,
&IID_ICaptureGraphBuilder2,
(void **)&pCaptureGraphBuilder'x');
pCaptureGraphBuilder'x'->SetFiltergraph(pGraphBuilder'x');
```

The webcam filter `pSrc'x'` is added to the filter graph manager `pGraphBuilder'x'` and the webcam filter pins are enumerated, so the output pin can be located for

rendering. Before rendering the function `InitVideoWindow'x'()` is called and the video window dimensions are defined. The video can now be displayed by using the `pMediaControl'x'->Run()` command:

```
pGraphBuilder'x'->AddFilter(pSrc'x', L"Video Capture");
pSrc'x'->EnumPins(&pEnum'x');
pEnum1->Reset();
pEnum1->Next(1, &m_pCamOutPin'x', NULL);
if( ! InitVideoWindow3(hVidWnd3, pWidth, pHeight) )
pGraphBuilder'x'->Render(m_pCamOutPin'x');
pMediaControl'x'->Run();
```

This process is completed for each webcam selected by the user.

5.3.6 Real-Time Edge Detection

The edge detection process was implemented from the code provided by Dr John Leis. The entire edged detection process has been compiled into a single header file (*.h), which is included in the main C source code.

The code is called when the user selects the Edge Detection button on the GUI. When enabled the function `ProcessFrame'x'()` is called and the current contents of the image buffers, `ImageBuffer` and `ImageBuffer1` are passed as arguments along with the image dimensions:

```
ProcessFrame'x'(ImageBuffer,ImageBuffer1,WebcamImageWidth,WebcamImageHeight);
```

The image data is converted into grayscale in the `ProcessFrameGrayscale()` function by extracting the red, green and blue components and scaling the individual colour channels:

```
BlueByte = *(pByteIn+0);
```

```
GreenByte = *(pByteIn+1);
RedByte = *(pByteIn+2);
```

The image is inverted and edges detected in both horizontal and vertical directions. The edge detection is achieved in the `ProcessFrameHorizEdge()` and `ProcessFrameVertEdge()` functions by scanning the pixel values for significatnt value changes:

```
ByteOut = 0;
if( abs(PixelValue - PrevPixelValue) > PixelThreshold )
{
    ByteOut = 255;
}
```

The user has the ability to adjust the `Pixel Threshold` value from 0 to 30 to increase or decrease the edged detection criteria.

5.3.7 Image Acquisition

Functionality was developed for the user to either independently or simultaneously grab video frames from the webcam video and store them disk in the Bitmap (*.BMP) format. The capture process was developed into the functions `GrabWebCamFrame'x'()` for each webcam.

VMR was required to capture the frame data from the streaming video by using the `GetCurrentImage` command that accessed the VMR video stream:

```
if(SUCCEEDED(hr = VMRpVidWin'x'->GetCurrentImage(&lpCurrImage)))
```

Each captured frame was processed and the necessary data collected to create a Bitmap file. The naming convention of Left, Centre and Right was selected for the three video windows. When the application is executed the numbering of images starts at '0' and

increments each time an image is captured, i.e. 'right0.bmp', 'right1.bmp'. If the program is closed and re-opened then the numbering is restarted at '0'.

The `CreateFile` and `WriteFile` calls were used for the creation and storage of the Bitmap file:

```
HANDLE WebcamFrame'x' = CreateFile(centerFileName, GENERIC_WRITE,
FILE_SHARE_READ, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);

if (WebcamFrame'x' == INVALID_HANDLE_VALUE)
return 0;

        dwSize = DibSize(pdib);
        hdr.bfType          = BFT_BITMAP;
        hdr.bfSize          = dwSize + sizeof(BITMAPFILEHEADER);
        hdr.bfReserved2     = 0;
        hdr.bfReserved2     = 0;
        hdr.bfOffBits       = (DWORD)sizeof(BITMAPFILEHEADER) +
 pdib->biSize + DibPaletteSize(pdib);

        WriteFile(WebcamFrame'x', (LPCVOID) &hdr, sizeof(BITMAPFILEHEADER),
&dwWritten, 0);
        WriteFile(WebcamFrame'x', (LPCVOID) pdib, dwSize, &dwWritten, 0);
        CloseHandle(WebcamFrame'x');
```

The images are stored in the same directory as the image capture platform.

5.3.8 Releasing DirectShow Objects

When the application is terminated it was necessary to correctly release all of the DirectShow objects in use. If this was not completed then other applications would not be able to access the webcams that were in use by the image capture platform.

The `CloseWebCamCapture()` function was created and includes all the necessary objects for disconnection or release. The function is called when the Stop button is selected or when the program is terminated by the Exit button or the windows close icon. The platform code below details the necessary objects that need to be handled for each webcam:

```
pMediaControl'x'->Stop();
m_pCamOutPin'x'->Disconnect();
m_pCamOutPin'x'->Release();
pSrc'x'->Release();
pMediaControl'x'->Release();
pMediaEvent'x'->Release();
InvalidateRect(hVidWnd'x', NULL, TRUE);
InvalidateRect(hGraWnd'x', NULL, TRUE);
```

The `InvalidateRect()` functions are included for clearing the video windows for the webcam and edge detection when the user selects the Stop button.

5.4 Library Development

The image capture library would allow additional accessibility through other applications or for developers working on projects requiring webcam access and image capture functionality. It was necessary then to develop the library in a suitable format that is widely compatible and easily accessed.

The Microsoft Windows DLL format was selected, as it would provide the greatest level of compatibility and support in the Microsoft Windows OS environment.

The task of developing the library was simplified due to the design requirements of:

- Access of up to two webcams for possible stereo processing.
- No GUI.

- No edge detection processing would be included.

The justification for the removal of the edge detection and GUI is that the application calling the library would be responsible for the interface and image processing.

The library was designed to 'wrap' the existing DirectShow components already developed as part of the image capture platform. The main requirement was a WINAPI entry point for other applications to reference. This was achieved with the `DllEntryPoint` function:

```
WINAPI DllEntryPoint();
```

The method and code for accessing the webcams and storing the captured images is identical to that in the image capture platform and described throughout this chapter.

5.5 Chapter Summary

The development stage for the image capture platform and library was the largest component of the project in regards to the skills and time required.

The image capture platform and library was developed in relation to the project objectives and allows for multiple webcams to operate and capture images. The interfacing process through the DirectShow API provided the largest developmental challenge, as the literature on the subject is loosely covered across numerous sources.

The coded examples provided by Dr John Leis provided a good explanation of the webcam access process through DirectShow.

Chapter 6

Experimental Approach and Testing of Image Capture and Processing

6.1 Chapter Overview

Two key stages of experimentation and testing are necessary to evaluate the suitability of webcams for stereo vision. This chapter will discuss the approaches taken for image capture and stereo processing.

The first stage involves testing the operation and functionality of the image capture platform in both binary and library form. This also involves the simultaneous capture of images from the connected webcams.

The second stage focuses on the image and stereo processing experimentation. This involves the application of algorithms from the Camera Calibration Toolbox for MATLAB for calibration and rectification, while the Dense Stereo Algorithm will be applied to the calibrated image pairs for stereo disparity mapping.

6.2 Image Capture Platform GUI Operation

The image capture platform GUI is designed to enumerate the first five devices connected to the computer system via the USB 2.0 interface. Testing was carried out on both test platforms with the same webcam configurations for consistency. The final design of the GUI can be seen in Figure 5.3.

An additional webcam was used for testing the device enumeration process, as the platform was designed to enumerate up to five devices. The image capture platform was tested with the webcam configurations in Table 6.1.

Test	Input
Number of webcams	1 - 5
Frames per second	10, 15, 20, 25, 30
Resolution	160 x 120, 320 x 240, 640 x 480, 800 x 600*, 1600 x 1200*
Format	RGB24, I420, YUY2 and MJPG
	*Note: The C200 webcam is limited to 640 x 480.

Table 6.1: Image capture platform webcam configuration for testing

Image capture was completed on the three selected devices. The output files have been predefined to output in bitmap (*.BMP) format. The captured images were then compared and evaluated for correct naming, format and resolution.

6.2.1 Edge Detection Processing

The edge detection processing was tested for correct operation. Edge detection operates on all connected devices with the option to disable the processing, as it can be processor intensive. Edge detection testing also included the adjustment of the pixel threshold (Figure 6.1) setting that influences the rate of change between pixels and therefore how an edge is determined.

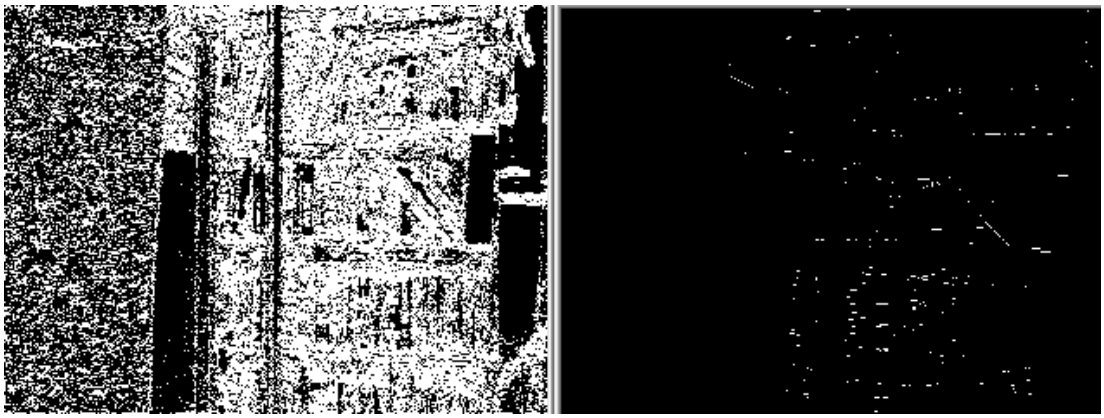

Figure 6.1: Testing of pixel threshold for edge detection processing

6.3 Image Capture Library Operation

The image capture solftware library was compiled into the DLL format supported by the Microsoft Windows OS. The testing process of the library involved:

- Access through MATLAB
- Access through a test program written in C

The library was designed to support up to two devices for streaming and image capture.

6.3.1 Access Through MATLAB

MATLAB was selected for a testing platform, as it is widely used and provides a large selection of processing operations for further image processing.

The testing process involved:

1. Loading the DLL file with the `loadlibrary(dll name)` command.
2. Accsss to the DLL functions was through the `calllib(dll name, library function name)` command.
3. The `Webcam1` and `Webcam2` functions were called to access the webcams.

4. The `GrabWebCamFrame1` and `GrabWebCamFrame2` functions were called to capture images from the video stream.

5. Access of the captured images was tested by loading and showing the image in the MATLAB workspace, by using the `imread(image name)` and `imshow(image name)` functions.

6. The library was unloaded once access was no longer required.

This testing process was completed on a single and dual webcam configurations. Figure 6.2 provides a screenshot of library output with dual webcams and a single captured image.

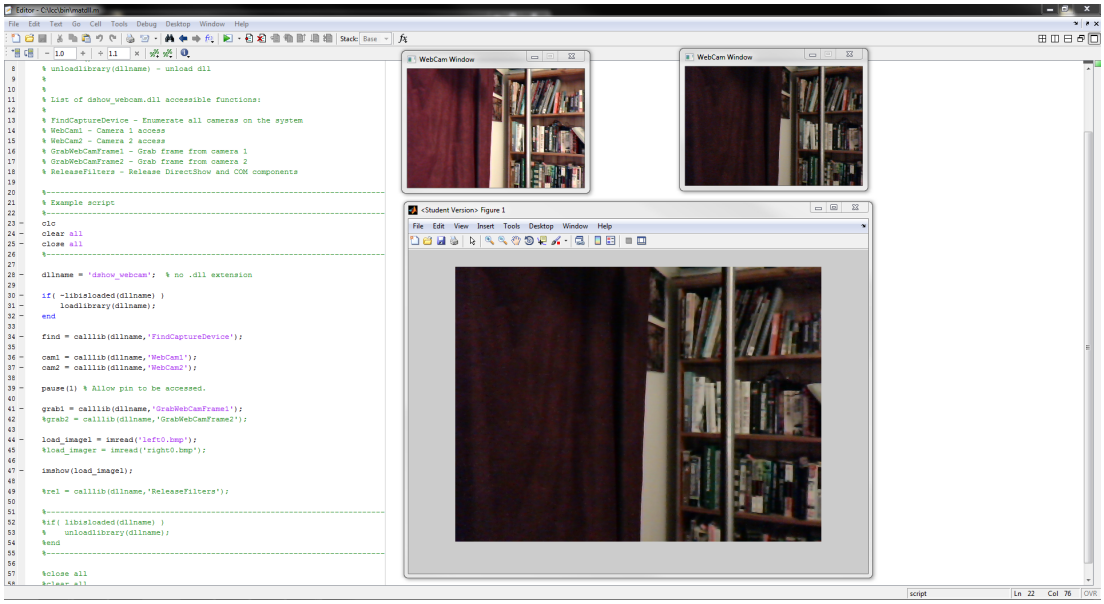

Figure 6.2: Testing of image capture library in MATLAB

6.3.2 Access Through C

The library is required to be accessed by other development projects and applications. A test program was written in C to evaluate and test the image capture library functionality.

The testing process involved:

1. Creating a basic Win32 application.
2. Loading the DLL file with the `LoadLibrary(dll name)` WINAPI command.
3. The required functions were called.
4. The `Webcam1` and `Webcam2` functions were called to access the webcams.
5. The `GrabWebCamFrame1` and `GrabWebCamFrame2` functions were called to capture images from the video stream and to save them to disk.
6. The library was unloaded unsing the `FreeLibrary` WINAPI command once access was no longer required.

This testing process was completed on a single and dual webcam configurations. Figure 6.3 provides a screenshot of library output with dual webcams.

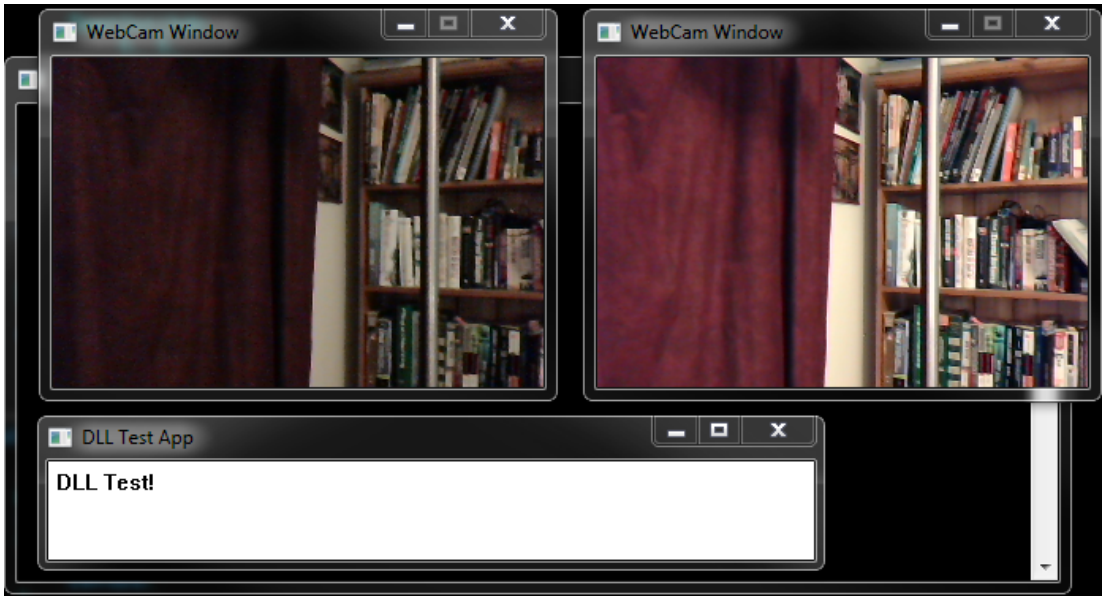

Figure 6.3: Testing of image capture library in C

6.4 Scene Capture and Stereo Processing

Experimentation of the stereo processing involved numerous considerations to ensure reliable results could be obtained. Considerations were made for:

- The type of scene and objects to be captured
- Physical webcam mounting
- Pre-processing of images including calibration and image rectification

Once these considerations were addressed, the stereo processing of image pairs could be completed for disparity mapping generation. The Camera Calibration Toolbox for MATLAB was used for the image processing stages, which included the calibration and image rectification, while the Dense Stereo algorithm developed by Abhijit Ogale was used for stereo processing.

6.4.1 Object Position and Scene Type

In order to provide a reliable set of output data for analysis, it was a necessity to experiment with scenes containing:

- Differences in object distance relative to the webcams
- A selection of objects that provide differences in shape, colour and texture.
- Shadows and object occlusions.
- Surfaces of a single colour or tone.
- Objects that provide horizontal and vertical edges.

From these requirements two scenes were selected that would include all of the preferred requirements:

- The Bookshelf
- The Living Room

6.4.2 Scene 1 - The Bookshelf

The Bookshelf scene was selected, as it provides object and edge variety over a relative short distance of 3.0 m . This reflects a common office based application of the webcam system and ensures a good level of disparity between the images.

Figure 6.4: Book shelf scene with points of interest for stereo processing

The points of interest in the scene include:

- A - Light Stand approximately 0.75 m from the webcam pair that provides a good vertical edge and partial occlusion of the ladder in the background.
- B - Book on stand approximately 1.2 m from the webcam pair.
- C - Partly occluded ladder and object leaning on ladder at a distance of 2.5 m from the webcam pair.
- D - Bookshelf containing sections of shadow above books.
- E - Poster on wall containing little surface change at distance of 3.0 m.

6.4.3 Scene 2 - The Living Room

The Living Room scene was selected for experimentation as it provides a longer overall distance of approximately 12.0 m from the webcams to the far wall. This will allow the stereo processing to be applied on image pairs that include small levels of disparity.

Figure 6.5: Living room scene with points of interest for stereo processing

The points of interest in the scene include:

- A - The large amount of surfaces with little change in geometry or tone.
- B - The even distribution of furniture over the scene length.
- C - High concentration points of light from the ceiling lights.

6.4.4 Webcam Mounting

The baseline distance while mounting the webcams (Figure 6.6) was tested for both webcam pairs at 50 - 150 mm. This value range was selected to provide a variation of disparity during testing, that also would allow the analysis of processor usage and stereo processing time.

During the mounting stage, care was taken to align the webcams accurately. This was achieved by viewing the live video stream of the scene and adjusting the alignment.

6.4.5 Webcam Calibration

Image pairs were captured from both pairs of webcams for the calibration process. A checkerboard pattern with a 30 mm grid spacing was used during the capture process to provide orientation and distance parameters for the Camera Calibration Toolbox. A

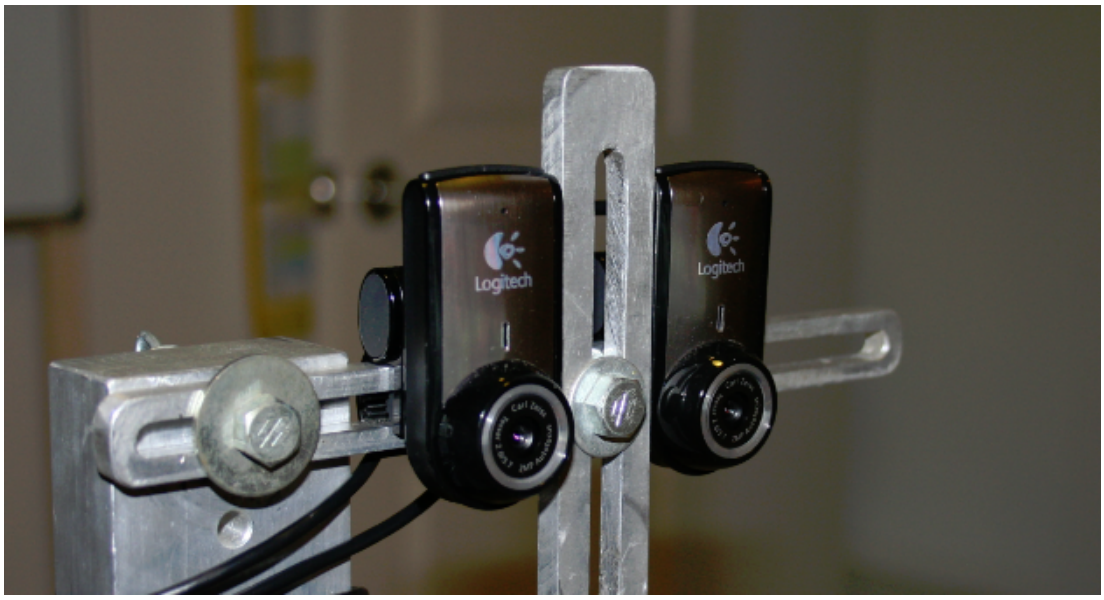

Figure 6.6: Webcam mounting configuration with a 50 mm baseline value

set of 10 and 20 image pairs (Figure 6.7) was taken for the comparison of calibration accuracy.

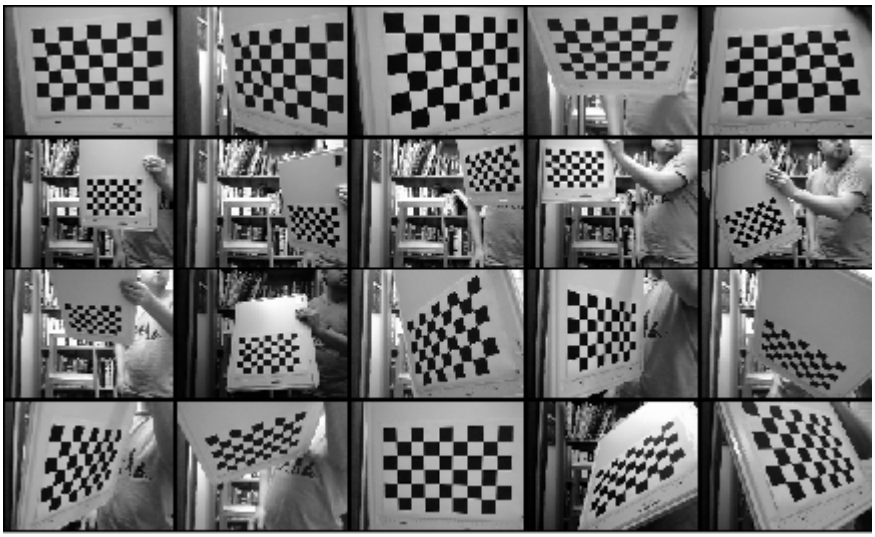

Figure 6.7: Collection of the left set of images for calibration

Calibration involved running the calibration module of the Camera Calibration Toolbox. This required the loading of the left and right image sets into MATLAB. Once the images were loaded, corner detection was carried out by manually selecting the checkerboard corners (Figure 6.8) for each image.

Once the corner detection process was completed for the left and right webcams the

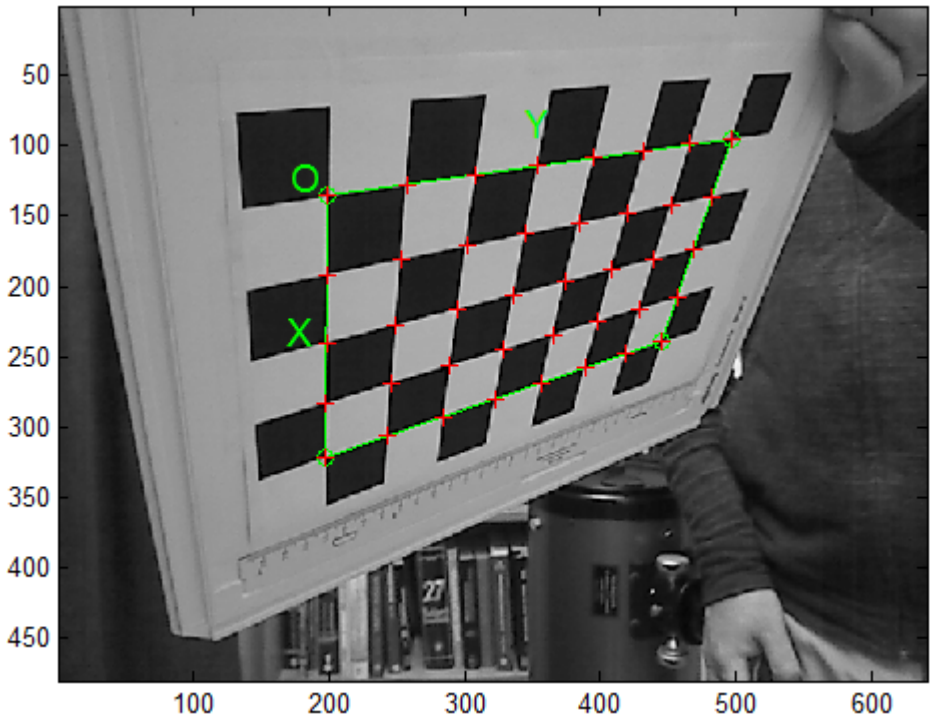

Figure 6.8: Manual selection of checkerboard corners

individual webcams were calibrated to compensate for any intrinsic defects that were found. These defects include:

- Radial lens distortion.
- Focal length compensation.
- Principal point compensation.

An example of the level of distortion in one of the Logitech C200 webcams can be seen in Figure 6.9.

After the webcams are individually calibrated for the correction of intrinsic properties, they then require stereo calibration to determine the extrinsic properties that relate to:

- The webcam to webcam relationship.
- Relationship between the webcams and the scene of interest.

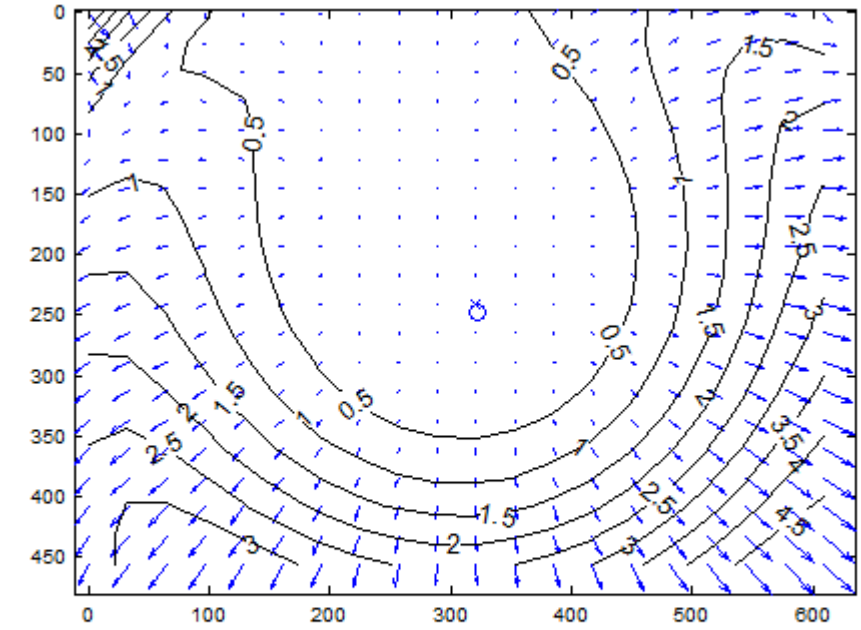

Figure 6.9: Lens distortion of the C200 webcam

The stereo calibration process involves analysing the individual webcam calibration properties and providing an estimate for the extrinsic parameters and the relative location of the right webcam with respect to the left webcam.

The left and right webcam checkerboard reference frames are then related to each other through rigid motion transformation. The final stereo calibration can be seen in Figure 6.10.

6.4.6 Image Rectification

The relationship between the two webcams was determined through the stereo calibration process in the Camera Calibration Toolbox. This allowed image rectification to be applied to the image pairs that had captured the test scenes. The image rectification removed any distortion and aligned each of the images from the pair along the horizontal image plane.

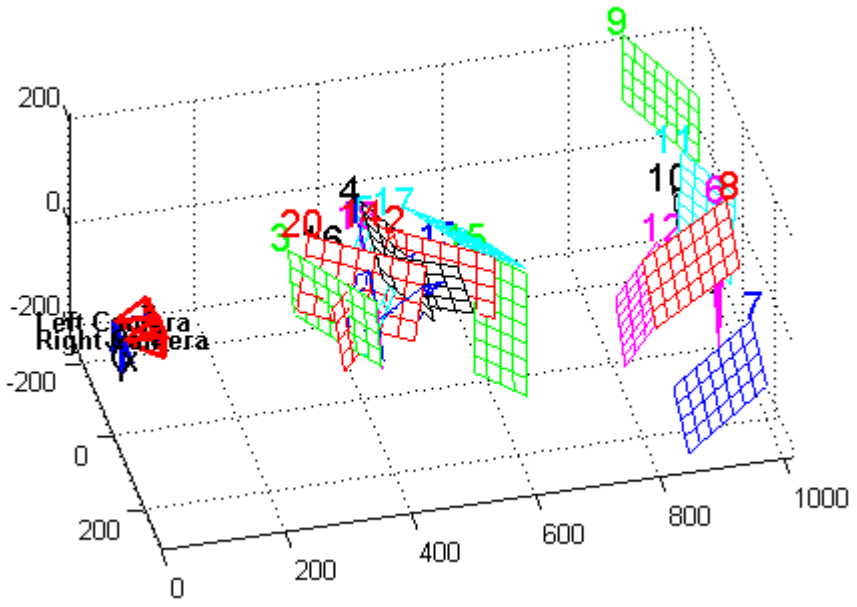

Figure 6.10: Stereo calibration result and extrinsic representation

This was necessary as the stereo processing was applied by scanning pixels along the horizontal image axis. Therefore any minor misalignment would result in less accurate disparity mapping.

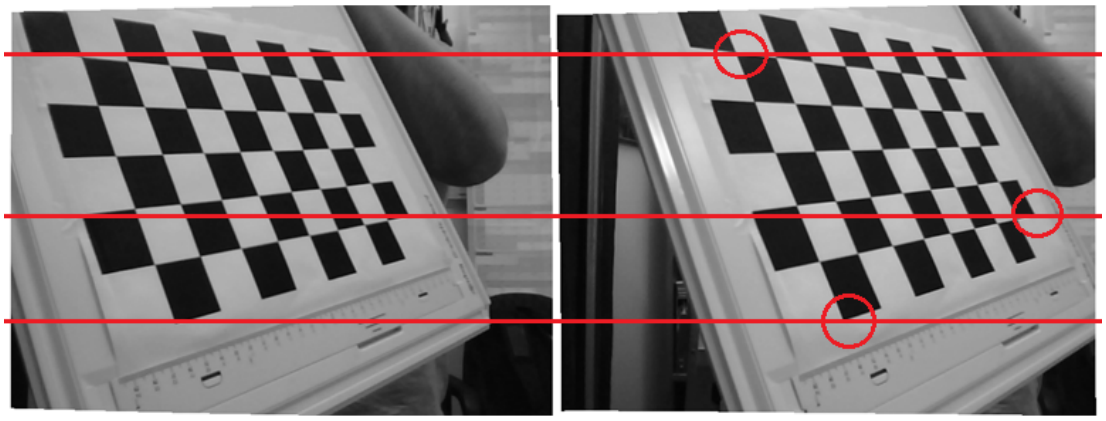

Figure 6.11: Image pair after removal of distortion and rectification applied

The alignment and removal of distortion during the rectification process is evident in Figure 6.11.

6.4.7 Stereo Processing

Stereo processing was applied to the calibrated image pairs for the generation of disparity maps. The Dense Stereo algorithm developed by Abhijit Ogale was selected for the experimentation, as it provides a robust and easy to use MATLAB operation.

The algorithm was first applied to the Tsukuba, Middlebury image set to enusre that the algorithm was operating correctly. The resulting disparity map can be seen in Figure 6.12.

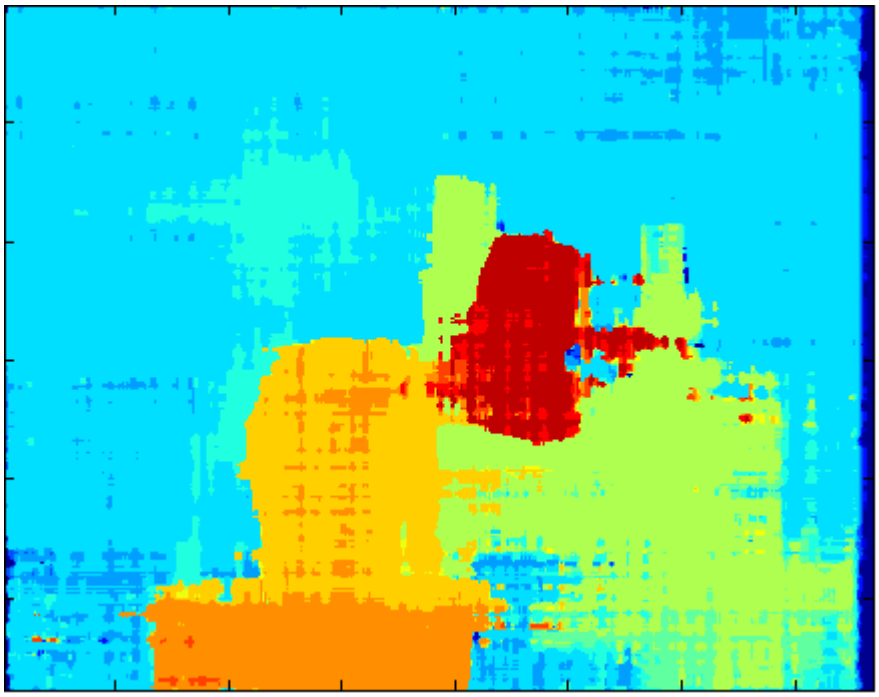

Figure 6.12: Test of the Middelbury benchmark image pair with the Dense Stereo algorithm

Stereo processing was then applied to the different test scenes with a set of different configurations. The configurations detailed in Table 6.2 ensured that a range of controlled experimental data could be collected and analysed.

The maximum selected resolution for stereo processing was 640 x 480. This was due the increase in stereo processing time, project time restraints and that the resolution is the most commonly supported with webcams.

The stereo processing was applied and results will be discussed in the next chapter.

Experiment	Configuration
Book shelf scene (calibrated)	C200 pair and C905 pair @ 640 x 480 resolution
Book shelf scene (un-calibrated)	C200 pair and C905 pair @ 640 x 480 resolution
Book shelf scene (calibrated)	C200 pair a@ 320 x 240 resolution
Living room scene	C905 pair @ 640 x 480 resolution

Table 6.2: Stereo processing configurations for experimentation

6.5 Chapter Summary

The experimental and testing approach has been discussed for both the image capture platform and stereo processing. The range of functionality testing on the image capture platform has provided a method for assessing the objective compliance and reviewing the overall platform design outcome.

The image processing testing and experimentation has covered the key aspects of stereo processing. The findings and results will be discussed in the next chapter.

Chapter 7

Results and Discussion

7.1 Chapter Overview

The results collected throughout the testing stage will be analysed to identify any problems and to determine the success of the project. The results will also be discussed in relation to the project objectives.

The functionality testing of the image capture platform and image capture library will be evaluated for correct operation. The image calibration, rectification and stereo processing testing will be analysed to determine the suitability of using consumer webcams for stereo vision.

7.2 Image Capture Platform Operation

Testing was completed on the image capture platform to evaluate the functionality and operation of the user controls and webcam access.

As detailed in Chapter 6, the image capture platform was tested on all three test platforms to ensure reliability and accessibility is achieved.

7.2.1 Enumeration of Devices

On execution the image capture platform was able to enumerate and list the first five webcams attached the test platform. This provided an easy method for selecting the preferred webcams. The first five webcams are identified and ordered by the Windows OS.

A minor issue was identified when operating multiple webcams of the same model, as the manufacturer identifies each webcam model identically. DirectShow collects this information during enumeration and displays it in the GUI drop down menus. This may require the user to run the selection process more than once to achieve the correct order of webcams. This issue does not impact on the operation of the platform.

7.2.2 Access and Operation of Devices

After the webcams were selected the webcam property dialog window was displayed for each selected device. The dialog window provides three properties for configuration, that vary depending on the webcam and driver in use:

- Frame Rate - in Frames Per Second.
- Color Space / Compression - YUY2, MJPG, RGB24 or I420.
- Output Size - resolution in pixels.

A combination of the three properties were selected during testing. Test platforms 1 and 3 operated the webcams through the Windows default UVC driver. This provided YUY2 and MJPG color space / compression options.

When using the YUY2 option the image capture platform was unable to operate more than two webcams simultaneously using the webcams default resolution and frame rate of 640 x 480 pixels and 30 FPS. By reducing the resolution or frame rate, it was possible to operate all three webcams. This issue was not webcam specific, instead it was determined to be driver based.

There were no issues in operating all three webcams with the MJPG mode. The webcams operated at all resolutions and allowable frame rates. The frame rates were limited on the C905 webcam, when a resolution greater than 1280 x 720 was selected. This limitation could not be removed.

The results for the test platforms 1 and 3 are provided in Table 7.1.

Test Platform 1 and 3			
Mode: YUY2			
	Webcam 1	Webcam 2	Webcam 3
Resolution	640 x 480	640 x 480	640 x 480
Frame Rate	30	30	30
Operating	Y	Y	N
Resolution	320 x 240	320 x 240	320 x 240
Frame Rate	30	30	30
Operating	Y	Y	Y
Resolution	640 x 480	640 x 480	640 x 480
Frame Rate	15	15	15
Operating	Y	Y	Y
Resolution	160 x 120	320 x 240	640 x 480
Frame Rate	15	15	15
Operating	Y	Y	Y
Mode: MJPG			
Resolution	ALL	ALL	ALL
Frame Rate	ALL	ALL	ALL
Operating	Y	Y	Y

Table 7.1: Test Platform 1 and 3 access and operation results and configuration

Test platform 2 used the manufactures driver for operation. This provided the color space / compression options of RGB24 and I420. An identical problem occurred in these modes, as did occur when YUY2 was selected on test platforms 1 and 3. The solution also required modifying the resolutions and frame rates.

The manufacture webcam driver was removed and testing completed with the Microsoft webcam driver. This provided correct operation in the MJPG mode.

The results for test platform 2 are provided in Table 7.2.

Test Platform 2			
Mode: RGB24 and I420			
	Webcam 1	Webcam 2	Webcam 3
Resolution	640 x 480	640 x 480	640 x 480
Frame Rate	30	30	30
Operating	Y	Y	N
Resolution	320 x 240	320 x 240	320 x 240
Frame Rate	30	30	30
Operating	Y	Y	Y
Resolution	640 x 480	640 x 480	640 x 480
Frame Rate	15	15	15
Operating	Y	Y	Y
Resolution	160 x 120	320 x 240	640 x 480
Frame Rate	15	15	15
Operating	Y	Y	Y
Mode: MJPG			
Resolution	ALL	ALL	ALL
Frame Rate	ALL	ALL	ALL
Operating	Y	Y	Y

Table 7.2: Test Platform 2 access and operation results and configuration

7.2.3 Image Capture and Image Output

The "Grab" function for the capture of image frames operated correctly on all the test platforms. At 30 FPS and at a resolution of 640 x 480 pixels, the worst case of time difference between the three simultaneous captures was 62 ms. At 30 fps it is possible to have up to 33 ms of difference between webcams, while additional time is required to capture and write the three images.

The writing to disk of the bitmap images operated correctly with the correct naming conventions and numbering used. On examining the images captured with the different color space / compression modes, there was no considerable difference in image quality.

7.2.4 Edge Detection Processing

The built-in edge detection processing was tested for operation and performance. The user control for enabling and disabling the detection process operated correctly. This functionality was included, so performance could be increased if the edge detection was not required.

Performance of the edge detection was evaluated on all three test platforms. Test platform 2 tested all four color space / compression modes, while platforms 1 and 3 tested only the YUY2 and MJPG modes, as only the Windows webcam driver was installed. Two common resolutions of 640 x 480 and 320 x 240 were chosen to determine the frame rate. The frame rate was calculated by monitoring the number of edge detection frames over a 60 second period and determining the second average.

The results for the edge detection performance are provided in Table 7.3.

Test Platform 2 - Frames Per Second				
Mode:	YUY2	MJPG	RGB24	I420
640 x 480	1.6	1.4	1.5	1.5
320 x 240	4.3	3.7	4.3	4.8
Test Platform 1 - Frames Per Second				
Mode:	YUY2	MJPG		
640 x 480	1.7	1.8		
320 x 240	4.2	3.7		
Test Platform 3 - Frames Per Second				
Mode:	YUY2	MJPG		
640 x 480	1.6	1.7		
320 x 240	4.2	3.6		

Table 7.3: Frame rate for the three test platforms

There was no significant difference in performance between the test platforms and compression modes.

7.3 Image Capture Library Operation

The image capture library DLL was tested in both the MATLAB and C environments. The methods for testing in each environment have been described in chapter 6. The test have provided confirmation that the library implementation is possible.

7.3.1 Access Through MATLAB

The DLL was loaded successfully into the MATLAB work environment. This allowed access to the `Webcam()` and `GrabWebcamFrame()` functions. The installed webcams were detected and enumerated, with the first two being used for the image capture process.

As only the first two installed webcams are used, it was necessary that the webcams were correctly installed in the corresponding USB ports. This was achieved by trial and error.

The webcam property dialog box was displayed for each selected device. As with the image caputre platform, the dialog window provides three properties for configuration, that vary depending on the webcam and driver in use:

- Frame Rate - in Frames Per Second.
- Color Space / Compression - YUY2, MJPG, RGB24 or I420.
- Output Size - resolution in pixels.

Testing was completed in the same manner as the image capture platform, however as only two webcams were used there was no issue with Color Space / Compression

modes. The webcams were also able to successfully run simultaneously at all supported resolutions and frame rates.

The images were correctly written to disk and were retrievable with `imread(image name)` and `imshow (image name)` functions (Figure 6.2).

7.3.2 Access Through C

Accessing the DLL through a C based application was completed with `LoadLibrary(dll name)` WINAPI commands. The process of selecting webcam properties and capturing images was identical to that used in the MATLAB environment, with the use of `Webcam()` and `GrabWebcamFrame()` functions.

Each webcam operated correctly and images were written to disk for further access and processing, depending on the developers requirements(Figure 6.3).

7.4 Scene Capture and Stereo Processing

The two selected scenes; The Bookshelf (Figure 6.4) and the Living Room (Figure 6.5) were processed in MATLAB with the Camera Calibration Toolbox and the Dense Stereo algorithm. The two scenes were tested with both pairs of webcams and with different configurations described in Chapter 6. The comparison of processing times and disparity values is provided in Table 7.4.

The benchmark refers to the Middlebury benchmark in Figure 6.12 that was used to determine that the algorithm was operating correctly. The disparity values were required for the stereo processing and relate to the estimated pixel disparity between both images in an image pair.

Webcam	Scene	Resolution	Calibrated/ Rectified	Processing Time (sec)	Disparity Value
C200	Bookshelf	640 x 480	Y	7.6	100
C200	Bookshelf	640 x 480	N	14.2	100
C200	Bookshelf	320 x 240	Y	1.2	100
C200	Bookshelf	320 x 240	N	1.8	100
C905	Living Room	640 x 480	Y	4.2	50
C905	Living Room	640 x 480	N	6.0	50
C905	Living Room	640 x 480	Y	1.8	10
C905	Living Room	640 x 480	N	2.6	10
C905	Bookshelf	640 x 480	Y	4.2	50
C905	Bookshelf	640 x 480	N	6.3	50
C905	Bookshelf	640 x 480	Y	14.2	200
C905	Bookshelf	640 x 480	N	33.5	200
Benchmark	N/A	N/A	N/A	0.9	10

Table 7.4: Comparison of processing times and disparity

7.4.1 Webcam Comparison and Image Resolution

Testing was carried out on the Bookshelf scene to evaluate the differences between the two webcam pairs. Each webcam pair has a difference in the field of view and therefore the amount of data captured is different. The C905 has an increased field of view (Figure 7.2) compared to the C200 (Figure 7.4).

The processed images with different fields of view do provide some differences in disparity mapping, however they provide similar object identification, edge detection and same types of errors produced that are discussed in section 7.4.3.

The Bookshelf scene was also captured at a reduced resolution of 320 x 240 pixels to determine if there is any significant difference in disparity mapping (Figure 7.1). The processing time (Table 7.4), object clarity and depth estimation was reduced compared to the larger image pairs captured at a resolution of 640 x 480 pixels.

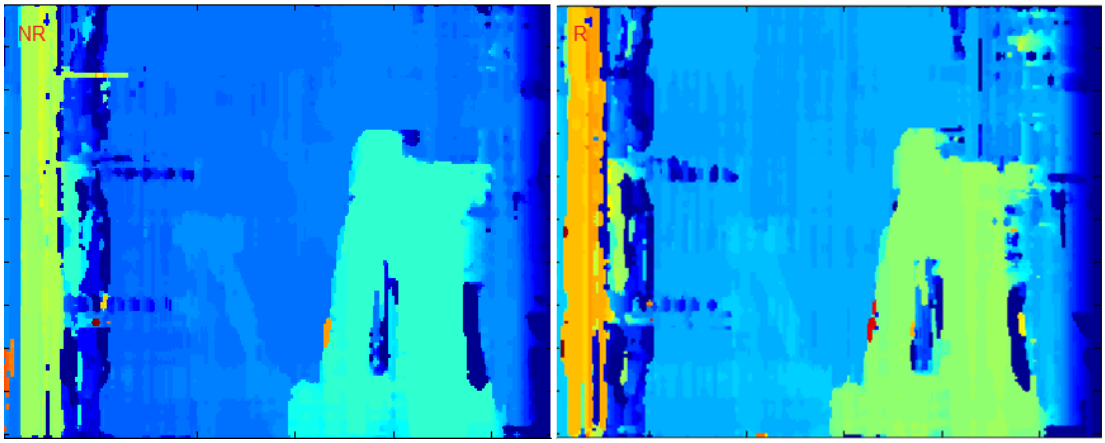

Figure 7.1: Bookshelf scene - C200 webcam at 320 x 240 pixels, non-rectified and rectified

The reduced accuracy was expected, as the amount of image data has halved. The disparity maps do still contain adequate information on object identification and depth estimation.

The use of different webcam models and different resolutions is important, as an outcome of the project is to provide a high level of accessibility to users.

7.4.2 Object Position and Scene Type

Each scene was selected for its variance in objects, surfaces, occlusions and distances. This was aimed at providing greater accuracy and precision in the resulting disparity map.

The Bookshelf scene provides key areas of importance in respect depth estimation and object detection. The scene includes a number of horizontal and vertical edges, that were included to test the accuracy of the stereo algorithm. The stereo algorithm scans the horizontal image axis for correlating points. It can be seen in Figure 7.4 that errors have been produced on the horizontal edges of the ladder steps and bookshelf. Opposite to this is the clarity of vertical edges in the scene, which are evident in the light stand and books in the foreground.

The differences in vertical and horizontal edges is less apparent in the Living Room scene (Figure 7.3), as it is almost completely composed of edges between 0 and 90

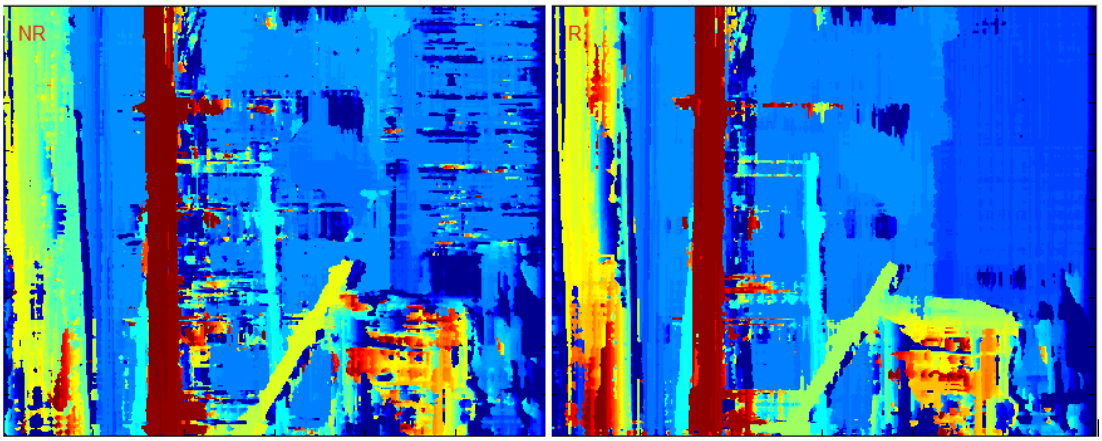

Figure 7.2: Bookshelf scene - C905 webcam at 640 x 480 pixels, non-rectified and rectified

degrees.

The inclusion of large surfaces, such as the far wall in Figure 7.4 provides little gradient change and has also produced inaccurate disparity mapping. This is a result of the algorithm not being able to distinguish any points of disparity, as no edges are found.

This is also apparent in the ceiling of the Living Room scene (Figure 7.3). The difference in this scene is the inclusion of ceiling lights that have provided some disparity for the stereo algorithm to process.

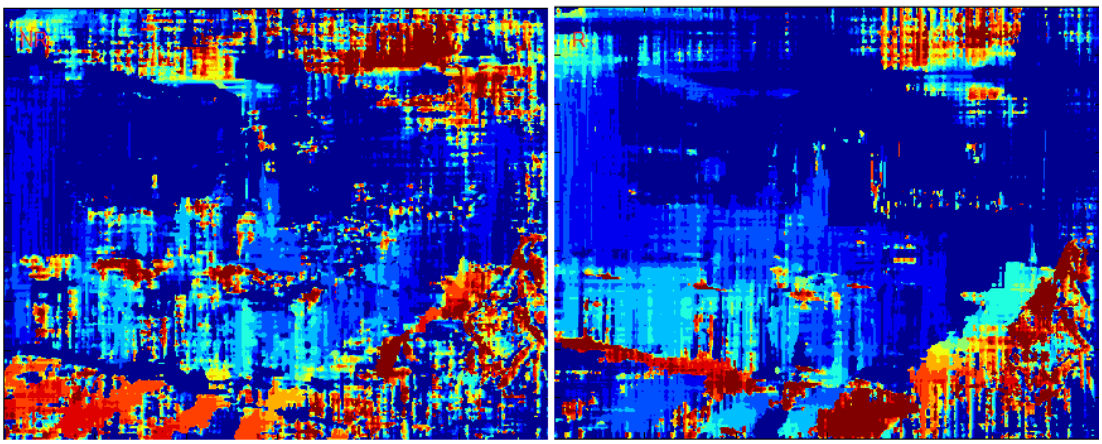

Figure 7.3: Living Room scene captured with the C905 webcam at 640 x 480 pixels, non-rectified and rectified

Each scene provides some level of object occlusion, as it is the nature of stereo vision that for disparity to occur there must be changes in object positions relative to the observer. The inclusion of the ladder being occluded by the light stand - A (Figure

6.4), provides a means for evaluating the resulting accuracy.

There is a small level of error produced, as the ladder starts to become visible behind the light stand (Figure 7.2). This has resulted in a small section of the ladder that appears closer in the foreground then it really is.

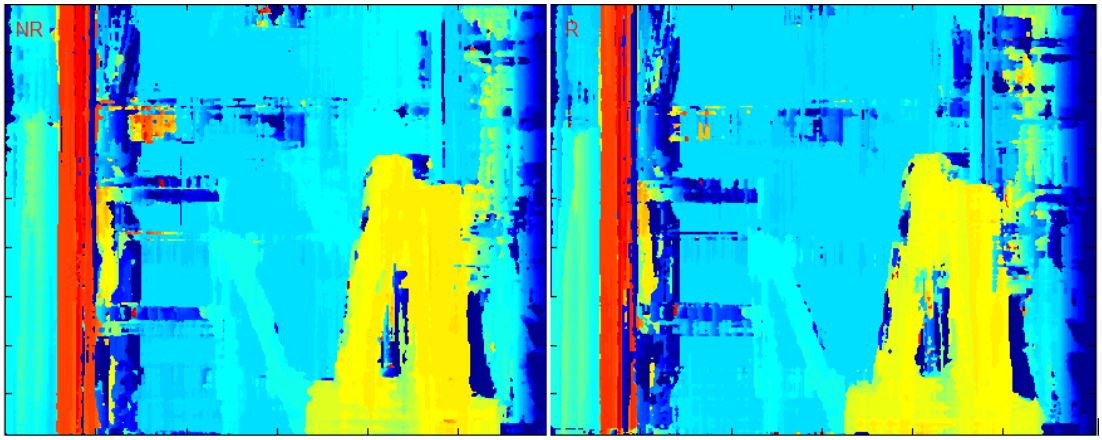

Figure 7.4: Bookshelf scene - C200 webcam at 640 x 480 pixels, non-rectified and rectified

Overall depth estimations from both scenes have resulted in a considerable difference between accuracy. This has occurred as a result of the overall scene distance and level of disparity between the image pairs. The accuracy of depth estimation is lower in the Living Room scene (Figure 7.3), as the disparity between the images is low when compared to the Bookshelf scene.

The relationship between distance and the level of disparity also affects the amount of processing time required. When the distance is increased, the level of disparity reduces and the processing time is reduced. The opposite occurs when the disparity increases as the distance reduces. This is also a contributing factor in the selection of a stereo baseline, as discussed in the next section.

7.4.3 The Effects of Webcam Mounting Configurations

Differences in disparity were evident between the 50 mm and 150 mm baselines that were selected for testing. The most significant differences in disparity accuracy were obtained from the Living Room scene (Figure 7.5). Due to the longer scene distance, there was a smaller amount of disparity between the left and right images. This provided

less accurate disparity at a baseline of 50 mm, however accuracy improved at 150 mm.

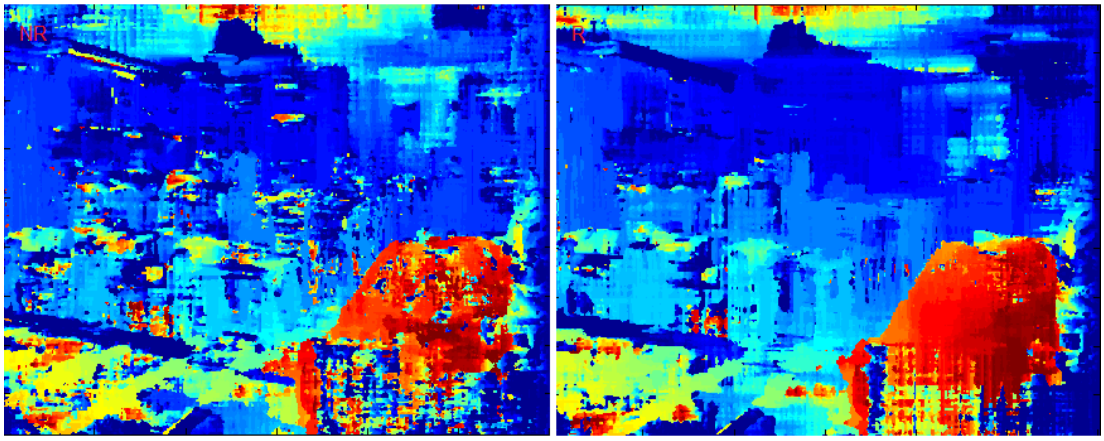

Figure 7.5: Living Room scene - C905 webcam at 640 x 480 pixels, non-rectified and rectified and with a baseline of 150 mm

The most notable improvement in the Living Room scene with a 150 mm baseline is the estimation of depth of foreground objects. The ceiling light points still provide an area of uncertainty when processing, however it has improved with the larger baseline.

The depth estimation in the Bookshelf scene was improved with the 150 mm baseline, however the level of disparity between the image pairs with a 50 mm baseline was already significant due to the shorter scene distance. Improvements can be observed in the detection of foreground objects including the telescope tube (Figure 7.6). The depth estimation does degrade with background objects, as the ladder and bookshelf object are apparent in Figure 7.2, but become harder to distinguish in Figure 7.6. This was not expected, as it was predicted that the increase in disparity would allow the algorithm to distinguish objects with more clarity and accuracy.

The increased baseline does appear to improve depth information by increasing the disparity between images. The consequence of this is the increase in processing time for the image pairs, as the stereo algorithms requires a number of horizontal scans that is related to the amount of disparity. The processing times are located in Table 7.4 and show that the processing time has increase by approximately 2.5 times for an increase from 50 mm to 150 mm.

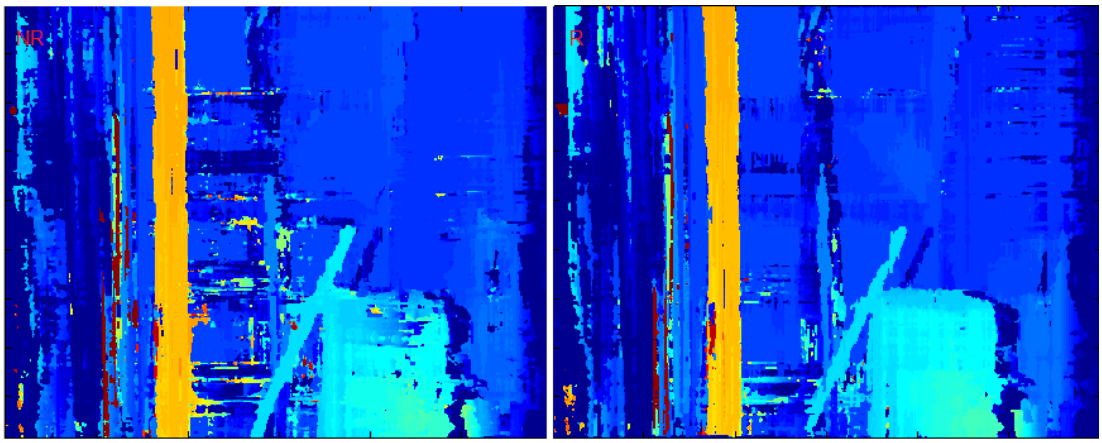

Figure 7.6: Book shelf scene - C905 webcam at 640 x 480 pixels, non-rectified and rectified and with a baseline of 150 mm

7.4.4 Webcam Calibration and Image Rectification Effects on Stereo Processing

The calibration process requires time and patience to achieve accurate and precise webcam calibration. The checkerboard technique implemented with the Camera Calibration Toolbox required approximately 20 minutes to obtain 20 image pairs and to apply corner detection for calibration.

Initial testing was completed with 10 image pairs, however this resulted in image rectification errors and poor disparity mapping. From experimentation it was found that at least 20 image pairs were required. Best results were obtained when the checkerboard was located in all four scene corners, at different angles and varying distances (Figure 6.7).

Image rectification was completed after the calibration process was completed, as the Camera Calibration Toolbox applied the collected calibration data to the image pairs. The figures (Figure 7.2) - (Figure 7.6) provide a comparison of the non- rectified and rectified images after stereo processing.

Differences in depth and object estimation can be observed in each scene. Figure 7.2 displays improvements in clarity of the objects in the scene. The detection of the edge of the bookshelf E (Figure 6.4) in relation to the wall is defined, while the amount of

error from the shadows D (Figure 6.4) in the bookshelf have reduced.

The difference is less apparent in Figure 7.1 that provides the disparity map for the lower resolution image pair. This result is expected, as the amount of image data for processing has reduced.

The Living Room scene (Figure 7.3) also displays the improvement of depth estimation and clarity from the calibration and rectification process. Improved object clarity can be observed in the objects B (Figure 6.5) while the large surfaces A (Figure 6.5) have less errors.

7.5 Chapter Summary

The image capture platform and image capture library have been tested and results analysed. Both the image capture techniques have been able to successfully complete all operation and functionality tasks outlined in the project objectives, with only minor issues that do not impact the overall operation.

The stereo vision processing and processes of calibration and image rectification have been evaluated. It has been found that consumer webcams can provide object and depth estimation with correct calibration and image rectification.

Chapter 8

Conclusions

8.1 Chapter Overview

The aim of this research project was to provide a solution for the simultaneous operation and image capture from consumer webcams and to investigate the suitability of webcams for stereo vision applications. The preceding chapters have provided details and discussion on the methodology and processes required for completing the projects aim.

After reviewing the work completed in this project it can be stated that the project aim has been met. The completion of this research project has identified areas of improvement and the opportunity for further work.

8.2 Achievement of Project Objectives

The project objectives have been assessed for their level of completion:

- **Research stereo and multiple camera vision systems including occlusion reduction and depth extraction techniques** - The field of stereo vision has been heavily researched over the years and therefore there is a large quantity

of existing literature on the subject. Chapter 2 provides information on the common approaches to applying stereo vision and the associated problems that occur due to the approaches. The research findings have been further examined in Chapters 6 and 7 and evaluated in relation to implementation of webcams as capture devices.

- **Research the feasibility of operating two or more USB webcam devices simultaneously on a single personal computer** - The operation of two or more webcams on a desktop computer system was research and discussed in Chapter 2. The findings identified that the operation is feasible and that the DirectShow API would provide the software layer for accessing the webcams. The implementation of the API is discussed in Chapter 5.

- **Design of a software access and control system for multiple webcam image acquisition and edge detection** - The design and development of the image capture platform and library was achieved. The methodology and approach to completing this objective was detailed in Chapter 3. The developmental process was covered in Chapter 5, while testing and results were discussed in Chapters 6 and 7.

- **Research calibration and stereo disparity processing techniques and the feasibility of applying them to the image acquisition system** - Existing literature on the subject was thoroughly researched and discussed in Chapter 2. It was identified that Camera Calibration Toolbox and Dense Stereo algorithm would be able to be applied to the captured image data for the evaluation of the webcams suitability for stereo vision. The testing and results of this are provided in Chapters 6 and 7.

- **Analyse captured image data for edge detection and surface processing accuracy** - The analysis of the captured images after stereo processing was completed and discussed in Chapter 7. The resulting images were examined for errors that may have occurred during the stereo vision process. It was identified that the images did contain some of the anomalies associated with stereo vision processing, including occlusion of objects and incorrect horizontal edge detection.

- **Design and implement image acquisition library into Dynamic-link li-**

brary format for access by other applications - The developmental and testing of the image capture library was achieved. The methodology and approach to completing this objective was detailed in Chapter 3. The development approach was discussed in Chapter 5, while testing and results are provided in Chapters 6 and 7.

Additional objectives that would be commenced if time and resources permit:

- **Investigate the feasibility of implementing the software system on multiple OS platforms** - Research was completed and discussed in Chapter 2 concerning the feasibility of using the image capture platform and library on other non-Windows based OS's. The Linux and Apple OS were investigated and it was found they can support the operation of multiple webcams through their OS specific API's.

- **Extend application research into infrared cameras and game based controllers** - Existing literature on the subject was researched and discussed in Chapter 2. Even though the use of infrared gaming controllers for motion tracking is a relatively new approach, there already has been a considerable amount of research and development into the application of stereo vision. The technology does appear to be well suited for accurate and robust stereo vision, while providing a low-cost hardware platform.

- **Design of software functions for multiple webcam calibration and stereo disparity processing** - Ideally the image capture platform would include built-in calibration and stereo processing capabilities similar to the edge detection capability. This has not been realised, as time limitations did not allow for further research and development to be undertaken. The scope of developing calibration and stereo processes is also large enough to be considered as an independent research project.

8.3 Shortcomings and Possible Improvements

The majority of the outlined objectives were successfully completed; however areas of improvement have been identified. The impact from the shortcomings have not impacted heavily on the overall success of the project, instead they would provide extra functionality and operation to the final image capture platform and library.

The GUI for the image capture platform is capable of enumerating up to five devices, however only the first three are able to be accessed. The ability for the user to select more than three webcams and also allowing them to control each output window would allow greater flexibility in controlling the webcams. A similar approach could also be applied to the image capture library that would increase the number of webcams accessible to more than two.

The captured images can only be written to disk in the BMP file format. This function could be further expanded to support other widely used image formats, including JPEG. The capture and writing process is also limited to writing the files to the root directory of the application. Improvements can be made to allow the user to select the storage location for the captured images, by implementing a basic file manager process.

It would also be of use to store the images into a memory buffer for quick retrieval and processing, instead of retrieval from the disk location. This capability would allow for other processing operations to be performed more efficiently and possibly in real-time.

The development of in-built calibration and stereo processing was outlined as an additional project objective, however it was not completed. By developing and implementing the processes into the image capture platform, it would provide a stand-alone application for image capture and stereo processing. This could be achieved by designing new algorithms or implementing existing algorithms for calibration and stereo processing. The algorithms could then be included in the image capture platform, much like the edge detection or developed into a library.

The resources involved in designing and implementing the algorithms would be too large to be identified as additional objectives within a research project, instead they

should be considered as further work.

8.4 Further Work

The findings and results from this project have identified key areas of further research and development work. The processing times for stereo vision provided in Chapter 7.4.1 have revealed that the realisation of a real-time system for stereo processing will require improvements in efficiency.

This may be possible with the included support for multi-core CPU's or even processing through the GPU. These techniques are becoming common, as most desktop computer systems include a dual or quad core CPU configuration, which would be ideal for processor intensive tasks, including stereo vision.

Research and development could be undertaken in the area of automatic calibration. As it was found that accurate manual calibration took approximately 20 minutes to complete, an approach for automation would be beneficial.

8.5 Final Conclusion

This project has shown that it is possible to operate multiple low-cost consumer web-cams simultaneously on a common desktop computer system. It has also been identified that the consumer webcam is capable of being implemented into a stereo vision system, if the correct calibration and image rectification processes have been completed.

The implication of these findings will allow an increased level of accessibility into the field of stereo vision research and development.

References

Appple (2011), 'Quicktime Overview', http://developer.apple.com/library/mac/#documentation/QuickTime/RM/Fundamentals/QTOverview/QTOverview_Document/QuickTimeOverview.html#//apple_ref/doc/uid/TP30000992-CH1g-QuickTimeOverview. [Online; accessed 3 July-2011].

Axelson, J. (2009), *USB Complete: The Developer's Guide*, 4th edn, Lakeview Research.

Ball, E. & Taschuk, G. (2011), 'Reverse Engineering the Kinect Stereo Algorithm', www.sccs.swarthmore.edu/users/13/gtaschuk/Kinect-proj3.pdf. [Online; accessed 17 July-2011].

Bhatti, A. & Nahavandi, S. (2008), 'Stereo correspondence estimation using multi-wavelets scale-space representation-based multiresolution analysis.', *Cybernetics and Systems* pp. 641–665.

Bradski, D. G. R. & Kaehler, A. (2008), *Learning opencv, 1st edition*, first edn, O'Reilly Media, Inc.

Carmody, T. (2010), 'How Motion Detection Works in Xbox Kinect', http://www.wired.com/gadgetlab/2010/11/tonights-release-xbox-kinect-how-does-it-work/. [Online; accessed 17 July-2011].

Castejon, C Blanco, D. M. (2009), 'Friendly interface to learn stereovision theory', *Computer Applications in Engineering Education* **17**, 180–186.

Chen, X. & Davis, J. (2000), Camera placement considering occlusion for robust motion capture, Technical report.

Cuypers, T Van, T. L. S. F. Y. (n.d.), 'STEREOWIISION: STEREO VISION WITH WIIMOTES'. [Online; accessed 7 August-2011].

Davies, E. R. (2004), *Machine Vision: Theory, Algorithms, Practicalities*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Engineers Australia (2011*a*), 'Code of Ethics', `http://engineersaustralia.org.au/about-us/corporate-reports/corporate-reports-home.cfm#ETHICS`. [Online; accessed 9 May-2011].

Engineers Australia (2011*b*), 'Sustainability Resources', `http://www.engineersaustralia.org.au/colleges/environmental/activities/sustainability/sustainability`. [Online; accessed 12 May-2011].

Forsyth, D. A. & Ponce, J. (2002), *Computer Vision: A Modern Approach*, Prentice Hall Professional Technical Reference.

Hay, S., Newman, J. & Harle, R. (2008), 'Optical tracking using commodity hardware', *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on* pp. 159–160.

House, B. C. & Nickels, K. (2006), 'Increased automation in stereo camera calibration techniques', *ournal of Systemics, Cybernetics and Informatics* **4**.

J. R. Asensio, J. M. M. M. & Montano, L. (1998), Navigation among obstacles by the cooperation of trinocular stereo vision system and laser rangefinder, Madrid, Spain, pp. 456 – 461.

Julesz, B. (1964), 'Binocular depth perception without familiarity cues', *Science* **145**, 356–362.

Lee, J. C. (2008), 'Hacking the nintendo wii remote', *IEEE Pervasive Computing* **7**, 39–45.

LinuxTV (2009), 'Standard Image Formats', `http://linuxtv.org/downloads/v4l-dvb-apis/ch02s03.html`. [Online; accessed 23 June-2011].

Medioni, G. & Kang, S. B. (2004), *Emerging Topics in Computer Vision*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Microsoft (2011*a*), 'Introduction to Directshow', `http://msdn.microsoft.com/en-us/library/windows/desktop/dd390351`. [Online; accessed 10 May-2011].

Microsoft (2011*b*), 'Video for Windows', `http://msdn.microsoft.com/en-us/library/windows/desktop/dd757708(v=vs.85).aspx`. [Online; accessed 21 June-2011].

Microsoft (2011*c*), 'What's New for Media Foundation', `http://msdn.microsoft.com/en-us/library/windows/desktop/bb970511(v=vs.85).aspx`. [Online; accessed 21 June-2011].

Mubarak, S. (1997), *Fundamentals of Computer Vision*, University of Central Florida, Orlando, FL.

Murphy, C., Lindquist, D., Rynning, A. M., Cecil, T., Leavitt, S. & Chang, M. L. (2007), 'Low-cost stereo vision on an fpga', *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on* **0**, 333–334.

Narasimha, R. (2010), 'Depth recovery from stereo matching using couple random fields', `http://perception.inrialpes.fr/Publications/2010/Nar10`.

Net Market Share (2011), 'Market Share for Mobile and Desktop', `http://www.netmarketshare.com/`. [Online; accessed 23 June-2011].

Pesce, M. D. (2002), *Programming Microsoft DirectShow for Digital Video, Television, and DVD*, Microsoft Press, Redmond, WA, USA.

R. Guerchouche, F. C. (2008), Camera calibration methods evaluation procedure for images rectification and 3d reconstruction, *in* 'Proceedings 16th International Conference in Central Europe on Computer Graphics', pp. 205–210.

Scharstein, D. & Szeliski, R. (2001), 'A taxonomy and evaluation of dense two-frame stereo correspondence algorithms', *INTERNATIONAL JOURNAL OF COMPUTER VISION* **47**, 7–42.

Ubuntu (2011), 'UVC Linux Driver', `https://help.ubuntu.com/community/UVC`. [Online; accessed 24 June-2011].

USQ (2011), *ENG4111 Research Project Reference Book: study book*, University of Southern Queensland, Toowoomba, QLD.

Zhang, Z. (2000), 'A flexible new technique for camera calibration', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**, 1330–1334.

Zitnick, C. & Kanade, T. (1999), A cooperative algorithm for stereo matching and occlusion detection, Technical Report CMU-RI-TR-99-35, Robotics Institute, Pittsburgh, PA.

Appendix A

Project Specification

ENG 4111/2 (or ENG8002) Research Project

Project Specification

For:	**Adam Cox**
Topic:	Stereo Vision with USB Webcams
Supervisor:	J.Leis
Program:	Computer Systems Engineering
Sponsorship:	Faculty of Engineering & Surveying
Project Aim:	To create a low cost, real time multiple webcam vision system with an investigation into the feasibility of providing stereo vision capabilities and a software platform for future research and development projects.

1. Research stereo and multi camera vision systems including occlusion reduction and depth extraction techniques.
2. Research the feasibility of operating two or more USB webcam devices simultaneously on a single desktop system.
3. Design of a software access and control system for multiple webcam image acquisition and edge detection.
4. Research calibration and stereo disparity processing techniques and the feasibility of applying them to the image acquisition system.
5. Analyse captured image data for edge detection and surface processing accuracy.
6. Design and implement image acquisition library into Dynamic-link library (DLL) format for access by other applications.

As time and resources permit:

1. Investigate the feasibility of implementing the software system on multiple OS platforms.
2. Extend application research into infrared cameras and game based controllers.
3. Design of software functions for multiple webcam calibration and stereo disparity processing.

Agreed:

Student Name:	Adam Cox
Date:	04/08/11
Supervisor Name:	John Leis
Date:	04/08/11
Examiner/Co-Examiner:	
Date:	

Appendix B

Project Timeline

	Task	From	To
1	**Research**	**01/03/11**	**12/08/11**
1.1	C in Windows	28/03/11	15/04/11
1.2	LCC configuration	28/03/11	21/04/11
1.3	Directshow programming in Windows	09/04/11	27/05/11
1.4	Webcam access in Non-Windows OS	01/03/11	28/04/11
1.5	DLL programming in Windows	01/03/11	28/04/11
1.6	Stereo Vision Techniques and Background	18/04/11	03/06/11
1.7	Calibration and Image Rectification	06/06/11	29/07/11
1.8	Infrared Gaming Controllers	18/07/11	12/08/11
2	**Develop Image Capture Platform**	**25/04/11**	**19/08/11**
2.1	Access Webcam in Windows	25/04/11	23/05/11
2.2	Create Directshow filters	25/04/11	23/05/11
2.3	Develop Interface for Webcams	16/05/11	03/06/11
2.4	Enumerate Multiple Devices	23/05/11	22/07/11
2.5	Configure Directshow and Platform	23/05/11	19/08/11
3	**Test Image Capture Platform**	**22/08/11**	**09/09/11**
3.1	Operation on Test Platforms	22/08/11	09/09/11
4	**Stereo Processing and Testing**	**09/09/11**	**30/09/11**
4.1	Capture image pairs	09/09/11	10/09/11
4.2	Rectify image pairs	13/09/11	17/09/11
4.3	Edge Detection	20/09/11	24/09/11
4.2	Create Disparity Mapping	17/09/11	20/09/11
4.3	Analysis of Disparity Accuracy	17/09/11	30/09/11
5	**Create Image Capture DLL**	**19/09/11**	**07/10/11**
5.1	Access in MATLAB	19/09/11	28/09/11
5.2	Access in C	26/09/11	07/10/11
6	**Test DLL**	**28/09/11**	**10/10/11**
6.1	Operation in MATLAB	28/09/11	05/10/11
6.2	Operation in C	04/10/11	10/10/11
7	**Dissertation**	**18/07/11**	**24/10/11**

Appendix C

Source Code Listings

C.1 Source - dshow_webcam.c

Main image capture platform GUI source code:

```
//------------------------------------------------------------------------------
//
//  Streams from up to 3 capture devices simultaneously using Microsoft Directshow.
//  Up to 5 devices are enumerated for access.
//
//------------------------------------------------------------------------------

#define WIN32_LEAN_AND_MEAN
#define STRICT

#pragma comment(lib,"user32.lib")
#pragma comment(lib,"gdi32.lib")
#pragma comment(lib,"shell32.lib")
#pragma comment(lib,"strmiids.lib")
#pragma comment(lib,"ole32.lib")
#pragma comment(lib,"amstrmid.lib")
#pragma comment(lib,"oleaut32.lib")
#pragma comment(lib,"uuid.lib")
#pragma comment(lib,"quartz.lib")


#include <windows.h>
#include <dshow.h> // Link with strmiids.lib and quartz.lib
#include <stdio.h>
#include <stddef.h>
#include <shlobj.h>
#include <string.h>
#include <ocidl.h>
#include "edge_detection.h" // Edge detection code provided by John Leis.

//==============================================================
// Function Prototypes
//==============================================================
int FindCaptureDevice(void);

int InitWebCamCapture1(HWND hVidWnd1, int *pWidth, int *pHeight);
int InitWebCamCapture2(HWND hVidWnd2, int *pWidth, int *pHeight);
int InitWebCamCapture3(HWND hVidWnd3, int *pWidth, int *pHeight);

int InitVideoWindow1(HWND hVidWnd1, int *pWidth, int *pHeight);
int InitVideoWindow2(HWND hVidWnd2, int *pWidth, int *pHeight);
int InitVideoWindow3(HWND hVidWnd3, int *pWidth, int *pHeight);

int InitializeWindowlessVMR1(HWND hVidWnd1);
int InitializeWindowlessVMR2(HWND hVidWnd2);
int InitializeWindowlessVMR3(HWND hVidWnd3);

int Convert24Image(BYTE *p32Img, BYTE *p24Img, DWORD dwSize32);

int StreamWebCamFrame1(unsigned char *pFrameOut, unsigned long FrameBufferLen,\
    int *pWidth, int *pHeight);
int StreamWebCamFrame2(unsigned char *pFrameOut, unsigned long FrameBufferLen,\
    int *pWidth, int *pHeight);
int StreamWebCamFrame3(unsigned char *pFrameOut, unsigned long FrameBufferLen,\
    int *pWidth, int *pHeight);

int StopWebCamCapture(void);
int PauseWebCamCapture(void);
int ResumeWebCamCapture(void);
int CloseWebCamCapture(void);

int GrabWebCamFrame1(void);
```

```
int GrabWebCamFrame2(void);
int GrabWebCamFrame3(void);

extern void     MouseMove(int x, int y);

extern int ProcessFrame1(unsigned char *pFrameIn, unsigned char *pFrameOut,
    int Width, int Height);
extern int ProcessFrame2(unsigned char *pFrameIn, unsigned char *pFrameOut,
    int Width, int Height);
extern int ProcessFrame3(unsigned char *pFrameIn, unsigned char *pFrameOut,
    int Width, int Height);

extern int ProcessFrameGrayscale(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height);
extern int ProcessFrameCopyBuf(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height);
extern int ProcessFrameInvert(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height);
extern int ProcessFrameVertEdge(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height);
extern int ProcessFrameHorizEdge(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height);
extern int ProcessFrameRedOnly(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height);
extern int ProcessFrameConv(unsigned char *pFrameIn,unsigned char *pFrameOut, \
    int Width, int Height, int xd, int yd);
extern int ProcessFrameRunLen(unsigned char *pFrameIn, unsigned char *pFrameOut, \
    int Width, int Height);

extern HBITMAP hCanvasBitmap;

//====================================================================
// Global Definitions
//====================================================================
IBaseFilter *pSrc1 = NULL; // Filters for 3 devices in use.
IBaseFilter *pSrc2 = NULL;
IBaseFilter *pSrc3 = NULL;

IBaseFilter *pEnumSrc1 = NULL; // Enumerate the first 5 devices.
IBaseFilter *pEnumSrc2 = NULL;
IBaseFilter *pEnumSrc3 = NULL;
IBaseFilter *pEnumSrc4 = NULL;
IBaseFilter *pEnumSrc5 = NULL;

IEnumPins *pEnum1 = NULL;
IEnumPins *pEnum2 = NULL;
IEnumPins *pEnum3 = NULL;

IGraphBuilder *pGraphBuilder1 = NULL;
IGraphBuilder *pGraphBuilder2 = NULL;
IGraphBuilder *pGraphBuilder3 = NULL;

IMediaControl *pMediaControl1 = NULL;
IMediaControl *pMediaControl2 = NULL;
IMediaControl *pMediaControl3 = NULL;

IMediaEventEx *pMediaEvent1 = NULL;
IMediaEventEx *pMediaEvent2 = NULL;
IMediaEventEx *pMediaEvent3 = NULL;

ICaptureGraphBuilder2 *pCaptureGraphBuilder1 = NULL;
ICaptureGraphBuilder2 *pCaptureGraphBuilder2 = NULL;
ICaptureGraphBuilder2 *pCaptureGraphBuilder3 = NULL;

IVMRWindowlessControl *VMRpVidWin1 = NULL;
IVMRWindowlessControl *VMRpVidWin2 = NULL;
IVMRWindowlessControl *VMRpVidWin3 = NULL;

IPin *m_pCamOutPin1 = NULL;
IPin *m_pCamOutPin2 = NULL;
IPin *m_pCamOutPin3 = NULL;
```

```
ISpecifyPropertyPages *pSpecPropPage1 = NULL;
ISpecifyPropertyPages *pSpecPropPage2 = NULL;
ISpecifyPropertyPages *pSpecPropPage3 = NULL;

//-----------------------------------------------------------------
//Bitmap definitions for header info.
typedef LPBITMAPINFOHEADER PDIB;
// Constants
#define BFT_BITMAP 0x4d42   /* 'BM' */
// Macros
#define DibNumColors(lpbi) ((lpbi)->biClrUsed == 0 && (lpbi)->biBitCount <= 8\
                           ? (int)(1 << (int)(lpbi)->biBitCount)\
                           : (int)(lpbi)->biClrUsed)

#define DibSize(lpbi)      ((lpbi)->biSize + (lpbi)->biSizeImage +\
                            (int)(lpbi)->biClrUsed * sizeof(RGBQUAD))
#define DibPaletteSize(lpbi)    (DibNumColors(lpbi) * sizeof(RGBQUAD))

//-----------------------------------------------------------------
// Webcam windows dimensions
#define WEBCAM_WINDOW_X      320
#define WEBCAM_WINDOW_Y      240

#define DEFAULT_CANVAS_WIDTH     320
#define DEFAULT_CANVAS_HEIGHT    240

static int WebcamImageWidth = DEFAULT_CANVAS_WIDTH;
static int WebcamImageHeight = DEFAULT_CANVAS_HEIGHT;
//-----------------------------------------------------------------
int cameraNum = 0; //Number of current enumerated capture devices.

int CurPos; // Trackbar Position
int DevMenuIndex = 0; // Index for menu order.

// bmp filename counters
int leftFileCount = 0;
int centerFileCount = 0;
int rightFileCount = 0;
//-----------------------------------------------------------------
// Camera status variables
int CameraActive1 = 0;
int CameraActive2 = 0;
int CameraActive3 = 0;

int WebCamRunning = 0;
int WebCamPaused = 0;
int WebCamInitialized = 0;

int ItemIndex1 = 0;
int ItemIndex2 = 0;
int ItemIndex3 = 0;
//-----------------------------------------------------------------
int edgeDetectOn = 0; // Edged detection control
static int PixelThreshold = 10; // Initial pixel threshold

// frame rate in ms
#define FRAMERATE        200
#define IDW_CAPWIN       1000
#define ID_TIMER         350

static TCHAR szAppName[] = TEXT ("Webcam_Capture");

static long nFrames = 0L;

unsigned long ImageBufferLen = 0L;

static BYTE *ImageBuffer = (BYTE *)NULL;
static BYTE *ImageBuffer1 = (BYTE *)NULL;

//=================================================================
```

```
// Define Windows
//=============================================================================
HINSTANCE    hInstance;

HWND     CreateMainWindow(HINSTANCE hInstance, HINSTANCE hPrevInstance, \
         LPSTR lpszCmdLine, int nWinMode);
HWND     CreateCaptureWindow(HWND hParentWnd);
HWND     CreateGrabWindow(HWND hParentWnd);

HWND     hWnd;
HWND     hMainWnd; // Main Window handle.

HWND     hPixTrackBar; // Pixel Threshold Trackbar Handle

HWND     hVidWnd1; // Camera Windows
HWND     hVidWnd2;
HWND     hVidWnd3;

HWND     hGraWnd1; // Frame Grab Windows
HWND     hGraWnd2;
HWND     hGraWnd3;

HWND     hWndComboBox1; // Combo Boxes
HWND     hWndComboBox2;
HWND     hWndComboBox3;

HWND     hTextImageDims;
HWND     hTextCursorPos;

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, \
    LPARAM lParam);
LRESULT CALLBACK WndProcCaptureWindow(HWND hwnd, UINT message, \
    WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndProcGrabWindow(HWND hwnd, UINT message, \
    WPARAM wParam, LPARAM lParam);

//=============================================================================
// Menu
//=============================================================================
#define ID_FILE_EXIT 9001
#define ID_DEVICE_LIST 9002
#define ID_ABOUT 9003

//=============================================================================
// Buttons
//=============================================================================
#define IDB_STARTBUTTON      110
#define IDB_STOPBUTTON       111
#define IDB_PAUSEBUTTON      112
#define IDB_RESUMEBUTTON     113
#define IDB_GRABLBUTTON      114
#define IDB_GRABCBUTTON      115
#define IDB_GRABRBUTTON      116
#define IDB_GRABABUTTON      117
#define IDB_EDBUTTON         118
#define IDB_EXITBUTTON       119
//=============================================================================
// Combo Boxes
//=============================================================================
#define IDC_COMBO1           120
#define IDC_COMBO2           121
#define IDC_COMBO3           122

//=============================================================================
// Track Bar
//=============================================================================
#define IDC_TRACK_PIX        130

//=============================================================================
// Main Window Menu
//=============================================================================
```

```
int BuildMenus(HWND hwnd)
{
    HMENU hMenu, hSubMenu;
    HICON hIcon, hIconSm;

    hMenu = CreateMenu();

    hSubMenu = CreatePopupMenu();

    AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu, "&File");
    AppendMenu(hSubMenu, MF_STRING, ID_FILE_EXIT, "E&xit");

    hSubMenu = CreatePopupMenu();
    AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu, "&About");
    AppendMenu(hSubMenu, MF_STRING, ID_ABOUT, "&About");

    SetMenu(hwnd, hMenu);

    return 1;
}
//=================================================================
// Find Capture Devices and Create Filters
//=================================================================

int FindCaptureDevice()
{
    // Remove Combo box contents.
    SendMessage(hWndComboBox1,CB_RESETCONTENT,0,0);
    SendMessage(hWndComboBox2,CB_RESETCONTENT,0,0);
    SendMessage(hWndComboBox3,CB_RESETCONTENT,0,0);

    SendMessage(hWndComboBox1,(UINT)CB_ADDSTRING,(WPARAM)0,(LPARAM)"Not Selected");
    SendMessage(hWndComboBox2,(UINT)CB_ADDSTRING,(WPARAM)0,(LPARAM)"Not Selected");
    SendMessage(hWndComboBox3,(UINT)CB_ADDSTRING,(WPARAM)0,(LPARAM)"Not Selected");

    SendMessage(hWndComboBox1,(UINT)CB_SETCURSEL,(WPARAM)0,(LPARAM)0);
    SendMessage(hWndComboBox2,(UINT)CB_SETCURSEL,(WPARAM)0,(LPARAM)0);
    SendMessage(hWndComboBox3,(UINT)CB_SETCURSEL,(WPARAM)0,(LPARAM)0);

    HRESULT hr = S_OK;
    cameraNum = 0;

    IMoniker *pMoniker= NULL;
    ICreateDevEnum *pDevEnum= NULL;
    IEnumMoniker *pClassEnum= NULL;
    IPropertyBag *pPropBag= NULL;

    // Create the system device enumerator
    hr = CoCreateInstance (&CLSID_SystemDeviceEnum,
                NULL,
                CLSCTX_INPROC,
                &IID_ICreateDevEnum,
                (void **) &pDevEnum);

    if (FAILED(hr))
    {
        MessageBox(0, TEXT("Device Enumeration Failed!"), 0, 0);
        return hr;
    }

    // Create an enumerator for the video capture devices
    if (SUCCEEDED(hr))
    {
        // Create an enumerator for the category.
        hr = pDevEnum->CreateClassEnumerator (&CLSID_VideoInputDeviceCategory, \
            &pClassEnum, 0);
        if (hr == S_FALSE)
        {
            hr = VFW_E_NOT_FOUND;  // The category is empty. Treat as an error.
        }
        pDevEnum->Release();
```

```
    if (FAILED(hr))
    {
        MessageBox(0, TEXT("No Devices Detected!"), 0, 0);
        return hr;
    }
}

if (SUCCEEDED(hr))
{
    // If there are no enumerators for the requested type, then
    // CreateClassEnumerator will succeed, but pClassEnum will be NULL.
    if (pClassEnum == NULL)
    {
        MessageBox(0, TEXT("Failed"), 0, 0);
        hr = E_FAIL;
        return hr;
    }
}

while (pClassEnum->Next(1, &pMoniker, NULL) == S_OK)
{

    HRESULT hr = pMoniker->BindToStorage(0, 0, &IID_IPropertyBag, \
        (void **)&pPropBag);
    if (FAILED(hr))
    {
        MessageBox(0, TEXT("Binding to Storage Failed"), 0, 0);
        pMoniker->Release();
        continue;
    }

    VARIANT var;
    var.vt = VT_BSTR;

    // Get description or friendly name.
    hr = pPropBag->Read(L"Description", &var, 0);
    if (FAILED(hr))
    {
        hr = pPropBag->Read(L"FriendlyName", &var, 0);

        char szName[256];
        // Convert BSTR
        WideCharToMultiByte(CP_ACP,0,var.bstrVal,-1,szName,256,0,0);
        // Store webcam name in combo box
        SendMessage(hWndComboBox1,(UINT)CB_ADDSTRING,(WPARAM)0,\
            (LPARAM)szName);
        SendMessage(hWndComboBox2,(UINT)CB_ADDSTRING,(WPARAM)0,\
            (LPARAM)szName);
        SendMessage(hWndComboBox3,(UINT)CB_ADDSTRING,(WPARAM)0,\
            (LPARAM)szName);
    }
    if (SUCCEEDED(hr))
    {
        VariantClear(&var);
        SysFreeString(var.bstrVal);
    }

    hr = pPropBag->Write(L"FriendlyName", &var);

    switch(cameraNum)
        {
        case 0:
            // Bind Moniker to a filter object
            pMoniker->BindToObject(0,0,&IID_IBaseFilter, \
                (void**)&pEnumSrc1);
            cameraNum ++;
            break;

        case 1:
            // Bind Moniker to a filter object
```

```
                pMoniker->BindToObject(0,0,&IID_IBaseFilter, \
                    (void**)&pEnumSrc2);
                cameraNum ++;
                break;

            case 2:
                // Bind Moniker to a filter object
                pMoniker->BindToObject(0,0,&IID_IBaseFilter, \
                    (void**)&pEnumSrc3);
                cameraNum ++;
                break;

            case 3:
                // Bind Moniker to a filter object
                pMoniker->BindToObject(0,0,&IID_IBaseFilter, \
                    (void**)&pEnumSrc4);
                cameraNum ++;
                break;

            case 4:
                // Bind Moniker to a filter object
                pMoniker->BindToObject(0,0,&IID_IBaseFilter, \
                    (void**)&pEnumSrc5);
                cameraNum ++;
                break;

            default:
                return hr;
            }

        // Release property bag and moniker
        pPropBag->Release();
        pMoniker->Release();

    }
    // release enumerator
    pClassEnum->Release();

    return 1;

}

//====================================================================
// Create Capture Device #1 Filter
//====================================================================
int     InitWebCamCapture1(HWND hVidWnd1, int *pWidth, int *pHeight)
{
    HRESULT hr;

    // Create the filter graph manager and query for interfaces.
    CoCreateInstance(&CLSID_FilterGraph, //Class ID for COM object
                NULL,
                CLSCTX_INPROC_SERVER,
                &IID_IGraphBuilder, // Interface ID
                (void **)&pGraphBuilder1); // Pointer back to FGM

    if( ! InitializeWindowlessVMR1(hVidWnd1) )
    {
        return 0;
    }

    // Media Control provides methods for flow of data through the
    // filter graph i.e. play, stop, pause...
    pGraphBuilder1->QueryInterface(&IID_IMediaControl, // Interface ID
                (void **)&pMediaControl1); // Pointer back to MC

    // Media Event provides methods for retrieving event notifications.
    pGraphBuilder1->QueryInterface(&IID_IMediaEventEx, // Interface ID
                (void **)&pMediaEvent1); // Pointer back to ME

    // Capture Graph Builder captures live video.
```

```
    CoCreateInstance(&CLSID_CaptureGraphBuilder2, //Class ID for COM object
                NULL,
                CLSCTX_INPROC,
                &IID_ICaptureGraphBuilder2, // Interface ID
                (void **)&pCaptureGraphBuilder1); // Pointer back to CGB2

    // Set the filter graph to capture graph.
    pCaptureGraphBuilder1->SetFiltergraph(pGraphBuilder1);

    // Attach the filter graph to capture graph.
    pGraphBuilder1->AddFilter(pSrc1, L"Video Capture");

    // Enumerate pins from Capture filter.
    pSrc1->EnumPins(&pEnum1);
    pEnum1->Reset();
    pEnum1->Next(1, &m_pCamOutPin1, NULL);

    // Pin Properties.
    hr = m_pCamOutPin1->QueryInterface(&IID_ISpecifyPropertyPages, \
        (void **)&pSpecPropPage1);
    if (SUCCEEDED(hr))
    {
        PIN_INFO PinInfo;
        m_pCamOutPin1->QueryPinInfo(&PinInfo);

        // Show the property page for user input.
        CAUUID caGUID;
        pSpecPropPage1->GetPages(&caGUID);
        OleCreatePropertyFrame(
            hMainWnd,
            0,
            0,
            L"Capture Device 1",
            1,
            (IUnknown **)&(m_pCamOutPin1),
            caGUID.cElems,
            caGUID.pElems,
            0,
            0,
        NULL);
        CoTaskMemFree(caGUID.pElems);
        PinInfo.pFilter->Release();
    }

    if( ! InitVideoWindow1(hVidWnd1, pWidth, pHeight) )
    {
        return 0;
    }

    hr = pGraphBuilder1->Render(m_pCamOutPin1);
    if (FAILED(hr))
    {
        return 0;
    }

    // Run live capture from device.
    hr = pMediaControl1->Run();

    pEnum1->Release();

    WebCamInitialized = 1;
    WebCamPaused = 0;

    return 1;
}

//=============================================================
// Create Capture Device #2 Filter
//=============================================================
int     InitWebCamCapture2(HWND hVidWnd2, int *pWidth, int *pHeight)
{
```

```
HRESULT hr;

// Create the filter graph manager and query for interfaces.
CoCreateInstance(&CLSID_FilterGraph, //Class ID for COM object
                 NULL,
                 CLSCTX_INPROC_SERVER,
                 &IID_IGraphBuilder, // Interface ID
                 (void **)&pGraphBuilder2); // Pointer back to FGM

if( ! InitializeWindowlessVMR2(hVidWnd2) )
{
    return 0;
}

// Media Control provides methods for flow of data through the filter
// graph i.e. play, stop, pause...
pGraphBuilder2->QueryInterface(&IID_IMediaControl, // Interface ID
                 (void **)&pMediaControl2); // Pointer back to MC

// Media Event provides methods for retrieving event notifications.
pGraphBuilder2->QueryInterface(&IID_IMediaEventEx, // Interface ID
                 (void **)&pMediaEvent2); // Pointer back to ME

// Capture Graph Builder captures live video.
CoCreateInstance(&CLSID_CaptureGraphBuilder2, //Class ID for COM object
                 NULL,
                 CLSCTX_INPROC,
                 &IID_ICaptureGraphBuilder2, // Interface ID
                 (void **)&pCaptureGraphBuilder2); // Pointer back to CGB2

// Set the filter graph to capture graph.
pCaptureGraphBuilder2->SetFiltergraph(pGraphBuilder2);

// Attach the filter graph to capture graph.
pGraphBuilder2->AddFilter(pSrc2, L"Video Capture");

// Enumerate pins from source base filter.
pSrc2->EnumPins(&pEnum2);
pEnum2->Reset();
pEnum2->Next(1, &m_pCamOutPin2, NULL);

// Pin Properties.
hr = m_pCamOutPin2->QueryInterface(&IID_ISpecifyPropertyPages, \
    (void **)&pSpecPropPage2);
if (SUCCEEDED(hr))
{

    PIN_INFO PinInfo;
    m_pCamOutPin2->QueryPinInfo(&PinInfo);

    // Show the property page for user input.
    CAUUID caGUID;
    pSpecPropPage2->GetPages(&caGUID);
    OleCreatePropertyFrame(
        hMainWnd,
        0,
        0,
        L"Capture Device 2",
        1,
        (IUnknown **)&(m_pCamOutPin2),
        caGUID.cElems,
        caGUID.pElems,
        0,
        0,
    NULL);
    CoTaskMemFree(caGUID.pElems);
    PinInfo.pFilter->Release();
}

if( ! InitVideoWindow2(hVidWnd2, pWidth, pHeight) )
{
```

```
        return 0;
    }

    hr = pGraphBuilder2->Render(m_pCamOutPin2);
    if (FAILED(hr))
    {
        return 0;
    }

    // Run live capture from device.
    hr = pMediaControl2->Run();

    pEnum2->Release();

    WebCamInitialized = 1;
    WebCamPaused = 0;

    return 1;
}

//=============================================================
// Create Capture Device #3 Filter
//=============================================================
int     InitWebCamCapture3(HWND hVidWnd3, int *pWidth, int *pHeight)
{
    HRESULT hr;

    // Create the filter graph manager and query for interfaces.
    CoCreateInstance(&CLSID_FilterGraph, //Class ID for COM object
                NULL,
                CLSCTX_INPROC_SERVER,
                &IID_IGraphBuilder, // Interface ID
                (void **)&pGraphBuilder3); // Pointer back to FGM

    if( ! InitializeWindowlessVMR3(hVidWnd3) )
    {
        return 0;
    }



    // Media Control provides methods for flow of data through the filter
    //graph i.e. play, stop, pause...
    pGraphBuilder3->QueryInterface(&IID_IMediaControl, // Interface ID
                (void **)&pMediaControl3); // Pointer back to MC

    // Media Event provides methods for retrieving event notifications.
    pGraphBuilder3->QueryInterface(&IID_IMediaEventEx, // Interface ID
                (void **)&pMediaEvent3); // Pointer back to ME

    // Capture Graph Builder captures live video.
    CoCreateInstance(&CLSID_CaptureGraphBuilder2, //Class ID for COM object
                NULL,
                CLSCTX_INPROC,
                &IID_ICaptureGraphBuilder2, // Interface ID
                (void **)&pCaptureGraphBuilder3); // Pointer back to CGB2

    // Set the filter graph to capture graph.
    pCaptureGraphBuilder3->SetFiltergraph(pGraphBuilder3);

    // Attach the filter graph to capture graph.
    pGraphBuilder3->AddFilter(pSrc3, L"Video_Capture");

    // Enumerate pins from Capture filter.
    pSrc3->EnumPins(&pEnum3);
    pEnum3->Reset();
    pEnum3->Next(1, &m_pCamOutPin3, NULL);

    // Pin Properties.
    hr = m_pCamOutPin3->QueryInterface(&IID_ISpecifyPropertyPages, \
        (void **)&pSpecPropPage3);
```

```
    if (SUCCEEDED(hr))
    {
        PIN_INFO PinInfo;
        m_pCamOutPin3->QueryPinInfo(&PinInfo);

        // Show the property page for user input.
        CAUUID caGUID;
        pSpecPropPage3->GetPages(&caGUID);
        OleCreatePropertyFrame(
            hMainWnd,
            0,
            0,
            L"Capture Device 3",
            1,
            (IUnknown **)&(m_pCamOutPin3),
            caGUID.cElems,
            caGUID.pElems,
            0,
            0,
        NULL);
        CoTaskMemFree(caGUID.pElems);
        PinInfo.pFilter->Release();
    }

    if( ! InitVideoWindow3(hVidWnd3, pWidth, pHeight) )
    {
        return 0;
    }

    hr = pGraphBuilder3->Render(m_pCamOutPin3);
    if (FAILED(hr))
    {
        return 0;
    }

    // Run live capture from device.
    hr = pMediaControl3->Run();

    pEnum3->Release();

    WebCamInitialized = 1;
    WebCamPaused = 0;

    return 1;
}


//=============================================================
// Webcam Controls
//=============================================================

// Pause the webcam via mediacontrol when pause button selected.
int     PauseWebCamCapture(void)
{
    HRESULT hr;

    if( ! WebCamInitialized )
    {
        return 0;
    }

    if( WebCamPaused )
    {
        return 0;
    }

    if(pMediaControl1)
    {
        hr = pMediaControl1->Pause();
        if( FAILED(hr) )
        {
```

```
            return 0;
        }
    }

    if(pMediaControl2)
    {
        hr = pMediaControl2->Pause();
        if( FAILED(hr) )
        {
            return 0;
        }
    }

    if(pMediaControl3)
    {
        hr = pMediaControl3->Pause();
        if( FAILED(hr) )
        {
            return 0;
        }
    }

    //-------------------------------------------------------------------------
    WebCamPaused = 1;
    //-------------------------------------------------------------------------

    return 0;
}
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Resume the webcam via mediacontrol when resume button is selected.
int      ResumeWebCamCapture(void)
{
    HRESULT hr;

    if( ! WebCamInitialized )
    {
        return 0;
    }

    if( ! WebCamPaused )
    {
        return 0;
    }

    if(pMediaControl1)
    {
       hr = pMediaControl1->Run();
        if( FAILED(hr) )
        {
            return 0;
        }
    }

     if(pMediaControl2)
    {
       hr = pMediaControl2->Run();
        if( FAILED(hr) )
        {
            return 0;
        }
    }

     if(pMediaControl3)
    {
       hr = pMediaControl3->Run();
        if( FAILED(hr) )
        {
            return 0;
        }
```

```
    }


    //-----------------------------------------------------------------------
    WebCamPaused = 0;
    //-----------------------------------------------------------------------

    return 0;
}
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// Close the application, release filters and webcams when exiting.
int     CloseWebCamCapture(void)
{
    HRESULT hr;

    if(CameraActive1 == 1)
    {
        pMediaControl1->Stop();
        m_pCamOutPin1->Disconnect();
        m_pCamOutPin1->Release();
        pSrc1->Release();
        pMediaControl1->Release();
        pMediaEvent1->Release();
        InvalidateRect(hVidWnd1, NULL, TRUE);
        InvalidateRect(hGraWnd1, NULL, TRUE);
    }

    if(CameraActive2 == 1)
    {
        pMediaControl2->Stop();
        m_pCamOutPin2->Disconnect();
        m_pCamOutPin2->Release();
        pSrc2->Release();
        pMediaControl2->Release();
        pMediaEvent2->Release();
        InvalidateRect(hVidWnd2, NULL, TRUE);
        InvalidateRect(hGraWnd2, NULL, TRUE);
    }

    if(CameraActive3 == 1)
    {
        pMediaControl3->Stop();
        m_pCamOutPin3->Disconnect();
        m_pCamOutPin3->Release();
        pSrc3->Release();
        pMediaControl3->Release();
        pMediaEvent3->Release();
        InvalidateRect(hVidWnd3, NULL, TRUE);
        InvalidateRect(hGraWnd3, NULL, TRUE);
    }

    WebCamInitialized = 0;
    WebCamRunning = 0;

    return 0;
}

//=========================================================================
// Windoless VMR #1 Function
//=========================================================================

// Windowless video control for video window placement.
int     InitializeWindowlessVMR1(HWND hVidWnd1)
{
    IBaseFilter        *pVmr1 = NULL;
    IVMRFilterConfig *pConfig1 = NULL;

    HRESULT hr;
```

```
    // Create the VMR and add it to the filter graph.
    hr = CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
            CLSCTX_INPROC, &IID_IBaseFilter, (void**)&pVmr1);

    if(FAILED(hr))
    {
        return 0;
    }

    hr = pGraphBuilder1->AddFilter(pVmr1, L"Video Mixing Renderer");
    if(FAILED(hr))
    {
        return 0;
    }

    // Set the rendering mode and number of streams.

    hr = pVmr1->QueryInterface(&IID_IVMRFilterConfig, (void**)&pConfig1);
    if(FAILED(hr))
    {
        return 0;
    }

    pConfig1->SetRenderingMode(VMRMode_Windowless);
    pConfig1->Release();

    hr = pVmr1->QueryInterface(&IID_IVMRWindowlessControl, (void**)&VMRpVidWin1);
    if(FAILED(hr))
    {
        return 0;
    }

    //Set VMR windowless output to windows handle
    VMRpVidWin1->SetVideoClippingWindow(hVidWnd1);

    //Release VMR control
    pVmr1->Release();

    return 1;
    }


//=============================================================
// Windoless VMR #2 Function
//=============================================================

// Windowless video control for video window placement.
int     InitializeWindowlessVMR2(HWND hVidWnd2)
{
    IBaseFilter         *pVmr2 = NULL;
    IVMRFilterConfig *pConfig2 = NULL;

    HRESULT hr;

    // Create the VMR and add it to the filter graph.
    hr = CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
            CLSCTX_INPROC, &IID_IBaseFilter, (void**)&pVmr2);

    if(FAILED(hr))
    {
        return 0;
    }

    hr = pGraphBuilder2->AddFilter(pVmr2, L"Video Mixing Renderer");
    if(FAILED(hr))
    {
        return 0;
    }

    // Set the rendering mode and number of streams.
```

```
    hr = pVmr2->QueryInterface(&IID_IVMRFilterConfig, (void**)&pConfig2);
    if(FAILED(hr))
    {
        return 0;
    }

    pConfig2->SetRenderingMode(VMRMode_Windowless);
    pConfig2->Release();

    hr = pVmr2->QueryInterface(&IID_IVMRWindowlessControl, (void**)&VMRpVidWin2);
    if(FAILED(hr))
    {
        return 0;
    }

    //Set VMR windowless output to windows handle
    VMRpVidWin2->SetVideoClippingWindow(hVidWnd2);

    //Release VMR control
    pVmr2->Release();

    return 1;
}


//=============================================================
// Windoless VMR #3 Function
//=============================================================

// Windowless video control for video window placement.
int     InitializeWindowlessVMR3(HWND hVidWnd3)
{
    IBaseFilter        *pVmr3 = NULL;
    IVMRFilterConfig *pConfig3 = NULL;

    HRESULT hr;

    // Create the VMR and add it to the filter graph.
    hr = CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
            CLSCTX_INPROC, &IID_IBaseFilter, (void**)&pVmr3);

    if(FAILED(hr))
    {
        return 0;
    }

    hr = pGraphBuilder3->AddFilter(pVmr3, L"Video Mixing Renderer");
    if(FAILED(hr))
    {
        return 0;
    }

    // Set the rendering mode and number of streams.

    hr = pVmr3->QueryInterface(&IID_IVMRFilterConfig, (void**)&pConfig3);
    if(FAILED(hr))
    {
        return 0;
    }

    pConfig3->SetRenderingMode(VMRMode_Windowless);
    pConfig3->Release();

    hr = pVmr3->QueryInterface(&IID_IVMRWindowlessControl, (void**)&VMRpVidWin3);
    if(FAILED(hr))
    {
        return 0;
    }

        //Set VMR windowless output to windows handle
    VMRpVidWin3->SetVideoClippingWindow(hVidWnd3);
```

```
    //Release VMR control
    pVmr3->Release();

    return 1;
}

//=============================================================
// Video Window #1 Function
//=============================================================

// Collect video window dimensions
int     InitVideoWindow1(HWND hVidWnd1, int *pWidth, int *pHeight)
{
    HRESULT hr;
    RECT rcDest;
    IAMStreamConfig  *pStreampConfig = NULL;
    IEnumMediaTypes *pEnumMediaType = NULL;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;
    VIDEOINFOHEADER *vidInfoHead = NULL;
    char    tmpbuf[1000];

    hr = m_pCamOutPin1->EnumMediaTypes(&pEnumMediaType);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    while(pEnumMediaType->Next(1, &pmt, 0) == S_OK)
    {
        if( memcmp((void *)& pmt->formattype, (void *)&FORMAT_VideoInfo, \
            sizeof(GUID)) == 0 )
        {

            vidInfoHead = (VIDEOINFOHEADER *)pmt->pbFormat;
            {
                pfnt = pmt;
                break;
            }
        }
    }

    pEnumMediaType->Release();

    hr = m_pCamOutPin1->QueryInterface(&IID_IAMStreamConfig, \
        (void **)&pStreampConfig);
    if(FAILED(hr))
    {
        return 0;
    }

    if( ! pfnt )
    {
        return 0;
    }

    hr = pStreampConfig->SetFormat(pfnt);

    CoTaskMemFree((void *)pfnt);

    hr = pStreampConfig->GetFormat(&pfnt);
    if(FAILED(hr))
    {
        return 0;
    }

    // Collect image width and height from video header.
    *pWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biWidth;
    *pHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biHeight;

    CoTaskMemFree((void *)pfnt);
```

```
    // Force preview to fixed size
    rcDest.left = 0;
    rcDest.top = 0;
    rcDest.right = 320;
    rcDest.bottom = 240;
    hr = VMRpVidWin1->SetVideoPosition(NULL, &rcDest);

    pStreampConfig->Release();

    return 1;
}

//=====================================================================
// Video Window #2 Function
//=====================================================================

// Collect video window dimensions
int     InitVideoWindow2(HWND hVidWnd2, int *pWidth, int *pHeight)
{
    HRESULT hr;
    RECT rcDest;
    IAMStreamConfig  *pStreampConfig = NULL;
    IEnumMediaTypes *pEnumMediaType = NULL;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;
    VIDEOINFOHEADER *vidInfoHead = NULL;
    char    tmpbuf[1000];

    hr = m_pCamOutPin2->EnumMediaTypes(&pEnumMediaType);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    while(pEnumMediaType->Next(1, &pmt, 0) == S_OK)
    {
        if( memcmp((void *)& pmt->formattype, (void *)&FORMAT_VideoInfo, \
            sizeof(GUID)) == 0 )
        {

            vidInfoHead = (VIDEOINFOHEADER *)pmt->pbFormat;
            {
                pfnt = pmt;
                break;
            }
        }
    }

    pEnumMediaType->Release();

    hr = m_pCamOutPin2->QueryInterface(&IID_IAMStreamConfig, \
        (void **)&pStreampConfig);
    if(FAILED(hr))
    {
        return 0;
    }

    if( ! pfnt )
    {
        return 0;
    }

    hr = pStreampConfig->SetFormat(pfnt);

    CoTaskMemFree((void *)pfnt);

    hr = pStreampConfig->GetFormat(&pfnt);
    if(FAILED(hr))
    {
        return 0;
    }
```

```
    // Collect image width and height from video header.
    *pWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biWidth;
    *pHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biHeight;

    CoTaskMemFree((void *)pfnt);

    // Force preview to fixed size
    rcDest.left = 0;
    rcDest.top = 0;
    rcDest.right = 320;
    rcDest.bottom = 240;
    hr = VMRpVidWin2->SetVideoPosition(NULL, &rcDest);

    pStreampConfig->Release();

    return 1;
}

//=====================================================================
// Video Window #3 Function
//=====================================================================

// Collect video window dimensions
int     InitVideoWindow3(HWND hVidWnd3, int *pWidth, int *pHeight)
{

    HRESULT hr;
    RECT rcDest;
    IAMStreamConfig  *pStreampConfig  = NULL;
    IEnumMediaTypes *pEnumMediaType  = NULL;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;
    VIDEOINFOHEADER *vidInfoHead = NULL;
    char    tmpbuf[1000];

    hr = m_pCamOutPin3->EnumMediaTypes(&pEnumMediaType);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    while(pEnumMediaType->Next(1, &pmt, 0) == S_OK)
    {
        if( memcmp((void *)& pmt->formattype, (void *)&FORMAT_VideoInfo, \
            sizeof(GUID)) == 0 )
        {

            vidInfoHead = (VIDEOINFOHEADER *)pmt->pbFormat;
            {
                pfnt = pmt;
                break;
            }
        }
    }

    pEnumMediaType->Release();

    hr = m_pCamOutPin3->QueryInterface(&IID_IAMStreamConfig, \
        (void **)&pStreampConfig);
    if(FAILED(hr))
    {
        return 0;
    }

    if( ! pfnt )
    {
        return 0;
    }

    hr = pStreampConfig->SetFormat(pfnt);
```

```
    CoTaskMemFree((void *)pfnt);

    hr = pStreampConfig->GetFormat(&pfnt);
    if(FAILED(hr))
    {
        return 0;
    }

    // Collect image width and height from video header.
    *pWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biWidth;
    *pHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biHeight;

    CoTaskMemFree((void *)pfnt);

    // Force preview to fixed size
    rcDest.left = 0;
    rcDest.top = 0;
    rcDest.right = 320;
    rcDest.bottom = 240;
    hr = VMRpVidWin3->SetVideoPosition(NULL, &rcDest);

    pStreampConfig->Release();

    return 1;
}

//====================================================================
// Frame Capture #1 Function
//====================================================================

// Collect frame from stream and convert to 24 bit if necessary
int      StreamWebCamFrame1(unsigned char *pFrameOut, unsigned long FrameBufferLen,\
     int *pWidth, int *pHeight)
{

    // Current frame buffer
    BYTE *lpCurrImage = NULL;
    LPBITMAPINFOHEADER  pdib;


    BYTE *pTemp32;
    unsigned long FrameSize24;
    unsigned long Height, Width;
    char     tmpbuf[1000];

    if( ! VMRpVidWin1    )
    {
        return 0;
    }

    // Collect frame from VMR stream
    if( VMRpVidWin1->GetCurrentImage(&lpCurrImage) != S_OK )
    {
        return 0;
    }

    pdib = (LPBITMAPINFOHEADER)lpCurrImage;
    pTemp32 = lpCurrImage + sizeof(BITMAPINFOHEADER);

    // Define video stream dimensions
    Height = pdib->biHeight;
    Width = pdib->biWidth;
    FrameSize24 = Width*Height*3L;


    *pWidth = Width;
    *pHeight = Height;

    if( FrameSize24 > FrameBufferLen )
    {
        CoTaskMemFree(lpCurrImage); //free the image
```

```
        return 0;
    }

    Convert24Image(pTemp32, pFrameOut, pdib->biSizeImage);

    CoTaskMemFree(lpCurrImage); //free the image

    return 1;
}

//====================================================================
// Frame Capture #2 Function
//====================================================================

// Collect frame from stream and convert to 24 bit if necessary
int     StreamWebCamFrame2(unsigned char *pFrameOut, unsigned long FrameBufferLen, \
    int *pWidth, int *pHeight)
{

    // Current frame buffer
    BYTE *lpCurrImage = NULL;
    LPBITMAPINFOHEADER  pdib;


    BYTE *pTemp32;
    unsigned long FrameSize24;
    unsigned long Height, Width;
    char    tmpbuf[1000];

    if( ! VMRpVidWin2   )
    {
        return 0;
    }

    // Collect frame from VMR stream
    if( VMRpVidWin2->GetCurrentImage(&lpCurrImage) != S_OK )
    {
        return 0;
    }

    pdib = (LPBITMAPINFOHEADER)lpCurrImage;
    pTemp32 = lpCurrImage + sizeof(BITMAPINFOHEADER);

    // Define video stream dimensions
    Height = pdib->biHeight;
    Width = pdib->biWidth;
    FrameSize24 = Width*Height*3L;

    *pWidth = Width;
    *pHeight = Height;

    if( FrameSize24 > FrameBufferLen )
    {
        CoTaskMemFree(lpCurrImage); //free the image
        return 0;
    }

    Convert24Image(pTemp32, pFrameOut, pdib->biSizeImage);

    CoTaskMemFree(lpCurrImage); //free the image

    return 1;
}

//====================================================================
// Frame Capture #3 Function
//====================================================================

// Collect frame from stream and convert to 24 bit if necessary
int     StreamWebCamFrame3(unsigned char *pFrameOut, unsigned long FrameBufferLen, \
    int *pWidth, int *pHeight)
```

```
{

    // Current frame buffer
    BYTE *lpCurrImage = NULL;
    LPBITMAPINFOHEADER  pdib;


    BYTE *pTemp32;
    unsigned long FrameSize24;
    unsigned long Height, Width;
    char     tmpbuf[1000];

    if( ! VMRpVidWin3    )
    {
        return 0;
    }

    // Collect frame from VMR stream
    if( VMRpVidWin3->GetCurrentImage(&lpCurrImage) != S_OK )
    {
        return 0;
    }

    pdib = (LPBITMAPINFOHEADER)lpCurrImage;
    pTemp32 = lpCurrImage + sizeof(BITMAPINFOHEADER);

    // Define video stream dimensions
    Height = pdib->biHeight;
    Width = pdib->biWidth;
    FrameSize24 = Width*Height*3L;

    *pWidth = Width;
    *pHeight = Height;

    if( FrameSize24 > FrameBufferLen )
    {
        CoTaskMemFree(lpCurrImage); //free the image
        return 0;
    }

    Convert24Image(pTemp32, pFrameOut, pdib->biSizeImage);

    CoTaskMemFree(lpCurrImage); //free the image


    return 1;
}

//=============================================================
// Convert Image Function
//=============================================================

// Convert frame bit count
int      Convert24Image(BYTE *p32Img, BYTE *p24Img, DWORD dwSize32)
{
    DWORD dwSize24;
    BYTE *pTemp,*ptr;
    DWORD index;
    unsigned char r, g, b;

    if( (! p32Img) || (! p24Img) || (dwSize32 == 0) )
    {
        return 0;
    }

    dwSize24=(dwSize32 * 3)/4;

    pTemp=p32Img;

    ptr=p24Img;
```

```
    for (index = 0; index < dwSize32/4 ; index++)
    {
        *ptr++ = *pTemp++;
        *ptr++ = *pTemp++;
        *ptr++ = *pTemp++;
        pTemp++;
    }
    return 1;
}

//=====================================================================
// Grab Frame From Webcam 1
//=====================================================================

// Grab frame and save as bitmap format
int GrabWebCamFrame1(void)
{
    HRESULT hr;
    BYTE* lpCurrImage = NULL;
    int ln = 0;

    if(CameraActive1 == 0 )
    {
        // Make sure devices are selected.
        MessageBox(0, TEXT("Frame grab failed on Device 1!"), 0, 0);
        CloseWebCamCapture();
        return 0;
    }

    if(SUCCEEDED(hr = VMRpVidWin1->GetCurrentImage(&lpCurrImage)))
    {

        BITMAPFILEHEADER    hdr;
        DWORD               dwSize, dwWritten;
        LPBITMAPINFOHEADER  pdib = (LPBITMAPINFOHEADER) lpCurrImage;

        const char *name = "left";
        char leftFileName[32] = {0};

        sprintf(leftFileName, "%s%d.bmp", name, leftFileCount);
        leftFileCount++;

        // Create handle and output bmp file
        HANDLE WebcamFrame1 = CreateFile(leftFileName, GENERIC_WRITE,
                              FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                              FILE_ATTRIBUTE_NORMAL, 0);

        if (WebcamFrame1 == INVALID_HANDLE_VALUE)
            return 0;

        // Initialize the bitmap header
        dwSize = DibSize(pdib);
        hdr.bfType          = BFT_BITMAP;
        hdr.bfSize          = dwSize + sizeof(BITMAPFILEHEADER);
        hdr.bfReserved1     = 0;
        hdr.bfReserved2     = 0;
        hdr.bfOffBits       = (DWORD)sizeof(BITMAPFILEHEADER) + \
                              pdib->biSize + DibPaletteSize(pdib);

        // Write the bitmap to the file
        WriteFile(WebcamFrame1, (LPCVOID) &hdr, sizeof(BITMAPFILEHEADER), \
            &dwWritten, 0);
        WriteFile(WebcamFrame1, (LPCVOID) pdib, dwSize, &dwWritten, 0);

        // Close the file
        CloseHandle(WebcamFrame1);

        // Free the image data returned from GetCurrentImage()
        CoTaskMemFree(lpCurrImage);
    }
    else
```

```
    {
        MessageBox(0, TEXT("Frame grab failed on Device 1!"), 0, 0);
        return 0;
    }

return 1;

}
//=====================================================================
// Grab Frame From Webcam 2
//=====================================================================

// Grab frame and save as bitmap format
int GrabWebCamFrame2(void)
{
    HRESULT hr;
    BYTE* lpCurrImage = NULL;

    if(CameraActive2 == 0 )
    {
        // Make sure devices are selected.
        MessageBox(0, TEXT("Frame grab failed on Device 2!"), 0, 0);
        CloseWebCamCapture();
        return 0;
    }

    if(SUCCEEDED(hr = VMRpVidWin2->GetCurrentImage(&lpCurrImage)))
    {



        BITMAPFILEHEADER    hdr;
        DWORD               dwSize, dwWritten;
        LPBITMAPINFOHEADER  pdib = (LPBITMAPINFOHEADER) lpCurrImage;

        const char *name = "center";
        char centerFileName[32] = {0};

        sprintf(centerFileName, "%s%d.bmp", name, centerFileCount);
        centerFileCount++;

        //Create handle and output bmp file
        HANDLE WebcamFrame2 = CreateFile(centerFileName, GENERIC_WRITE,
                              FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                              FILE_ATTRIBUTE_NORMAL, 0);

        if (WebcamFrame2 == INVALID_HANDLE_VALUE)
              return 0;

        // Initialize the bitmap header
        dwSize = DibSize(pdib);
        hdr.bfType          = BFT_BITMAP;
        hdr.bfSize          = dwSize + sizeof(BITMAPFILEHEADER);
        hdr.bfReserved2     = 0;
        hdr.bfReserved2     = 0;
        hdr.bfOffBits       = (DWORD)sizeof(BITMAPFILEHEADER) + \
                              pdib->biSize + DibPaletteSize(pdib);

        // Write the bitmap to the file
        WriteFile(WebcamFrame2, (LPCVOID) &hdr, sizeof(BITMAPFILEHEADER), \
            &dwWritten, 0);
        WriteFile(WebcamFrame2, (LPCVOID) pdib, dwSize, &dwWritten, 0);

        // Close the file
        CloseHandle(WebcamFrame2);

        // Free the image data returned from GetCurrentImage()
        CoTaskMemFree(lpCurrImage);
    }
    else
    {
```

```
        MessageBox(0, TEXT("Frame grab failed on Device 2!"), 0, 0);
        return 0;
    }

return 1;

}
//=====================================================================
// Grab Frame From Webcam 3
//=====================================================================

// Grab frame and save as bitmap format
int GrabWebCamFrame3(void)
{
    HRESULT hr;
    BYTE* lpCurrImage = NULL;

    if(CameraActive3 == 0 )
    {
        // Make sure devices are selected.
        MessageBox(0, TEXT("Frame grab failed on Device 3!"), 0, 0);
        CloseWebCamCapture();
        return 0;
    }

    if(SUCCEEDED(hr = VMRpVidWin3->GetCurrentImage(&lpCurrImage)))
    {

        BITMAPFILEHEADER    hdr;
        DWORD               dwSize, dwWritten;
        LPBITMAPINFOHEADER  pdib = (LPBITMAPINFOHEADER) lpCurrImage;

        const char *name = "right";
        char rightFileName[32] = {0};

        sprintf(rightFileName, "%s%d.bmp", name, rightFileCount);
        rightFileCount++;

        // Create handle and output bmp file
        HANDLE WebcamFrame3 = CreateFile(rightFileName, GENERIC_WRITE,
                                         FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                                         FILE_ATTRIBUTE_NORMAL, 0);

        if (WebcamFrame3 == INVALID_HANDLE_VALUE)
            return 0;

        // Initialize the bitmap header
        dwSize = DibSize(pdib);
        hdr.bfType          = BFT_BITMAP;
        hdr.bfSize          = dwSize + sizeof(BITMAPFILEHEADER);
        hdr.bfReserved2     = 0;
        hdr.bfReserved2     = 0;
        hdr.bfOffBits       = (DWORD)sizeof(BITMAPFILEHEADER) + pdib->biSize + \
                              DibPaletteSize(pdib);

        // Write the bitmap to the file
        WriteFile(WebcamFrame3, (LPCVOID) &hdr, sizeof(BITMAPFILEHEADER), \
            &dwWritten, 0);
        WriteFile(WebcamFrame3, (LPCVOID) pdib, dwSize, &dwWritten, 0);

        // Close the file
        CloseHandle(WebcamFrame3);

        // Free the image data returned from GetCurrentImage()
        CoTaskMemFree(lpCurrImage);
    }
    else
    {
        MessageBox(0, TEXT("Frame grab failed on Device 3!"), 0, 0);
        return 0;
    }
```

```
return 1;

}

//=============================================================================
// WinMain Entry
//=============================================================================

// Main API entry point.
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, \
    LPSTR lpszCmdLine, int nWinMode)
{

    MSG msg;

    // Create windows
    hMainWnd = CreateMainWindow(hInstance, hPrevInstance, lpszCmdLine, nWinMode);
    if( ! hMainWnd )
        return 0;

    hWnd = CreateCaptureWindow(hMainWnd);
    if(FAILED(CoInitializeEx(NULL, COINIT_MULTITHREADED)))
    {
        exit(1);
    }
    hWnd = CreateGrabWindow(hMainWnd);
    if(FAILED(CoInitializeEx(NULL, COINIT_MULTITHREADED)))
    {
        exit(1);
    }

    // Find devices attached to the system
    FindCaptureDevice();

    while( GetMessage(&msg, NULL, 0, 0) )
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}

//=============================================================================
// Create Main Window
//=============================================================================
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    RECT rect;

    int         ExtraWidth, ExtraHeight;
    int         nVirtKey;
    char        tmpbuf[1000];

    SCROLLINFO si;
    CurPos = 0;

    ZeroMemory(&si, sizeof(si));
    si.cbSize = sizeof(si);
    si.fMask  = SIF_RANGE | SIF_PAGE | SIF_TRACKPOS | SIF_POS;
    si.nMin   = 0;
    si.nMax   = 30;
    si.nPage  = 10;
    si.nPos = 10;
    si.nTrackPos = 10;

    SetScrollInfo(hPixTrackBar, SB_CTL, &si, TRUE);
```

```
switch( message )
{
    case WM_CREATE:
        BuildMenus(hwnd);
        return 0;

    case WM_COMMAND:

            // Select devices from combo boxes
            switch( HIWORD(wParam) )
            {

            case CBN_SELCHANGE:

                switch( LOWORD(wParam) )
                {
                    // Combo box 1
                    case IDC_COMBO1:

                    ItemIndex1 = SendMessage((HWND)hWndComboBox1,\
                    (UINT)CB_GETCURSEL,(WPARAM)0,(LPARAM)0);

                    switch(ItemIndex1)
                    {
                        case 0:
                            pSrc1 = NULL;
                            CameraActive1 = 0;
                        break;

                        case 1:
                            pSrc1 = pEnumSrc1;
                            CameraActive1 = 1;
                        break;

                        case 2:
                            pSrc1 = pEnumSrc2;
                            CameraActive1 = 1;
                        break;

                        case 3:
                            pSrc1 = pEnumSrc3;
                            CameraActive1 = 1;
                        break;

                        case 4:
                            pSrc1 = pEnumSrc4;
                            CameraActive1 = 1;
                        break;

                        case 5:
                            pSrc1 = pEnumSrc5;
                            CameraActive1 = 1;
                        break;
                        default:
                            break;
                    }

                    break;

                    // Combo box 2
                    case IDC_COMBO2:

                    ItemIndex2 = SendMessage((HWND)hWndComboBox2,\
                    (UINT)CB_GETCURSEL,(WPARAM)0,(LPARAM)0);

                    switch(ItemIndex2)
                    {
                        case 0:
                            pSrc2 = NULL;
                            CameraActive2 = 0;
                        break;
```

```
        case 1:
            pSrc2 = pEnumSrc1;
            CameraActive2 = 1;
        break;

        case 2:
            pSrc2 = pEnumSrc2;
            CameraActive2 = 1;
        break;

        case 3:
            pSrc2 = pEnumSrc3;
            CameraActive2 = 1;
        break;

        case 4:
            pSrc2 = pEnumSrc4;
            CameraActive2 = 1;
        break;

        case 5:
            pSrc2 = pEnumSrc5;
            CameraActive2 = 1;
        break;
        default:
            break;
    }

    break;

    // Combo box 3
    case IDC_COMBO3:

    ItemIndex3 = SendMessage((HWND)hWndComboBox3,\
    (UINT)CB_GETCURSEL,(WPARAM)0,(LPARAM)0);

    switch(ItemIndex3)
    {
        case 0:
            pSrc3 = NULL;
            CameraActive3 = 0;
        break;

        case 1:
            pSrc3 = pEnumSrc1;
            CameraActive3 = 1;
        break;

        case 2:
            pSrc3 = pEnumSrc2;
            CameraActive3 = 1;
        break;

        case 3:
            pSrc3 = pEnumSrc3;
            CameraActive3 = 1;
        break;

        case 4:
            pSrc3 = pEnumSrc4;
            CameraActive3 = 1;
        break;

        case 5:
            pSrc3 = pEnumSrc5;
            CameraActive3 = 1;
        break;
        default:
            break;
    }
```

```
                    break;

                }
                default:
                    break;
            }


            switch( LOWORD(wParam) )
            {

                case ID_FILE_EXIT:
                    PostMessage(hwnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
                    break;

                case ID_ABOUT:
                    MessageBox(0, TEXT("Mulitple Camera Access Platform.\
                    \nSupports up to 5 devices. \n\nCreated by Adam Cox 2011"), 0, 0);
                    break;

                //-------------------------------------------------
                // Button controls
                //-------------------------------------------------
                case IDB_STARTBUTTON:
                    //-------------------------------------------------
                    if( WebCamRunning )
                    {
                        // Webcams already running?
                        return 0;
                    }
                    //-------------------------------------------------
                    if(CameraActive1 == 0 &&  CameraActive2 == 0 && \
                            CameraActive3 == 0)
                    {
                        // Make sure devices are selected.
                        MessageBox(0, TEXT("No Devices Selected!"), 0, 0);
                        return 0;
                    }
                    //-------------------------------------------------
                    //Initiate cameras if attached and selected
                    //-------------------------------------------------
                    if(CameraActive1 == 1)
                    {
                        if( ! InitWebCamCapture1(hVidWnd1, &WebcamImageWidth,\
                         &WebcamImageHeight))
                        {
                            MessageBox(hwnd, "Capture Device #1 Could Not Initialise!",\
                            "WebCam", MB_OK);
                            return 0;
                        }
                    }

                    if(CameraActive2 == 1)
                    {
                        if( ! InitWebCamCapture2(hVidWnd2, &WebcamImageWidth,\
                        &WebcamImageHeight))
                        {
                            MessageBox(hwnd, "Capture Device #2 Could Not Initialise!",\
                            "WebCam", MB_OK);
                            return 0;
                        }
                    }

                    if(CameraActive3 == 1)
                    {
                        if( ! InitWebCamCapture3(hVidWnd3, &WebcamImageWidth,\
                        &WebcamImageHeight))
                        {
                            MessageBox(hwnd, "Capture Device #3 Could Not Initialise!",\
                            "WebCam", MB_OK);
                            return 0;
```

```
        }
    }
    //-----------------------------------------------------------------
    // Set image buffer length and create buffer
    //-----------------------------------------------------------------
    ImageBufferLen = WebcamImageWidth * WebcamImageHeight * 3L;

    ImageBuffer = (BYTE *)malloc(ImageBufferLen);
    if( ! ImageBuffer )
    {
        MessageBox(NULL, "Image buffer alloc failed", szAppName, MB_OK);
        break;
    }

    ImageBuffer1 = (BYTE *)malloc(ImageBufferLen);
    if( ! ImageBuffer1 )
    {
        MessageBox(NULL, "Image buffer1 alloc failed", szAppName, MB_OK);
        if( ImageBuffer )
        {
            free((void *)ImageBuffer);
            ImageBuffer = (unsigned char *)NULL;
        }
        break;
    }
    //-----------------------------------------------------------------

    //-------------------------------------------------
    // Set camera status
    //-------------------------------------------------
    WebCamRunning = 1;
    WebCamInitialized = 1;
    WebCamPaused = 0;
    //-------------------------------------------------

    //-------------------------------------------------
    sprintf(tmpbuf, "%d x %d", WebcamImageWidth, WebcamImageHeight);
    SetWindowText(hTextImageDims, tmpbuf);
    //-------------------------------------------------

#if 0
    //-------------------------------------------------
    if( StreamWebCamFrame1(ImageBuffer, ImageBufferLen, \
        &WebcamImageWidth, &WebcamImageHeight ) )
    if( StreamWebCamFrame2(ImageBuffer, ImageBufferLen, \
        &WebcamImageWidth, &WebcamImageHeight ) )
    if( StreamWebCamFrame3(ImageBuffer, ImageBufferLen, \
        &WebcamImageWidth, &WebcamImageHeight ) )
    //-------------------------------------------------
#endif

    //-------------------------------------------------
    // to resize image to actual driver size
    //-------------------------------------------------

    //-------------------------------------------------
    //GetWindowRect(hMainWnd, &rect);
    //sprintf(tmpbuf, "w=%d h=%d", rect.right, rect.bottom);
    //sprintf(tmpbuf, "l=%d t=%d", rect.left, rect.top);
    //MessageBox(hwnd, tmpbuf, "WebCam", MB_OK);
    //ExtraWidth = DEFAULT_CANVAS_WIDTH - rect.right;
    //ExtraHeight = DEFAULT_CANVAS_HEIGHT - rect.bottom;
    //-------------------------------------------------

    //-------------------------------------------------
    // resize to capture default
    //-------------------------------------------------

    //-------------------------------------------------
    /*GetWindowRect(hMainWnd, &rect);
    MoveWindow(hMainWnd,
```

```
                        rect.left, rect.top,
                        3*WebcamImageWidth + 200,
                        2*WebcamImageHeight + 120,
                        TRUE);

    MoveWindow(hVidWnd1,
                        140, 20,
                        WebcamImageWidth, WebcamImageHeight,
                        TRUE);

    MoveWindow(hGraWnd1,
                        140, 20 + WebcamImageHeight,
                        WebcamImageWidth, WebcamImageHeight,
                        TRUE);*/
    //---------------------------------------------------

    //---------------------------------------------------------------------
    SetTimer(hMainWnd, ID_TIMER, FRAMERATE, NULL);
    //---------------------------------------------------------------------

    //---------------------------------------------------------------------
    nFrames = 0L;
    //---------------------------------------------------------------------

    break;
//-------------------------------------------------

//-------------------------------------------------
case IDB_STOPBUTTON:
    //-------------------------------------------------
    // Reset pixel threshold and slider bar position
    //-------------------------------------------------
    PixelThreshold = 10;
    SendMessage(hPixTrackBar, TBM_SETPOS,(WPARAM) \
    TRUE,(LPARAM) 10);

    CloseWebCamCapture();

    //-------------------------------------------------
    break;

//-------------------------------------------------
case IDB_PAUSEBUTTON:
    //-------------------------------------------------
    if( ! WebCamRunning )
        return 0;

    if( WebCamPaused )
        return 0;
    //-------------------------------------------------

    PauseWebCamCapture();

    WebCamPaused = 1;

    break;
//-------------------------------------------------

//-------------------------------------------------
case IDB_RESUMEBUTTON:
    //-------------------------------------------------
    if( ! WebCamRunning )
        return 0;

    if( ! WebCamPaused )
        return 0;
    //-------------------------------------------------

    ResumeWebCamCapture();

    WebCamPaused = 0;
```

```
        break;
//---------------------------------------------------

//---------------------------------------------------
// Grab left frame
//---------------------------------------------------
case IDB_GRABLBUTTON:
        //-----------------------------------------------
        if( ! WebCamRunning )
            return 0;

        GrabWebCamFrame1();

        break;
//---------------------------------------------------

//---------------------------------------------------
// Grab center frame
//---------------------------------------------------
case IDB_GRABCBUTTON:
        //-----------------------------------------------
        if( ! WebCamRunning )
            return 0;

        GrabWebCamFrame2();

        break;
//---------------------------------------------------

//---------------------------------------------------
// Grab right frame
//---------------------------------------------------
case IDB_GRABRBUTTON:
        //-----------------------------------------------
        if( ! WebCamRunning )
            return 0;

        GrabWebCamFrame3();

        break;
//---------------------------------------------------

//---------------------------------------------------
// Grab all frames
//---------------------------------------------------
case IDB_GRABABUTTON:
        //-----------------------------------------------
        if( ! WebCamRunning )
            return 0;


        if(CameraActive1 == 1)
        {
            GrabWebCamFrame1();
        }

        if(CameraActive2 == 1)
        {
            GrabWebCamFrame2();
        }

        if(CameraActive3 == 1)
        {
            GrabWebCamFrame3();
        }


        break;
//---------------------------------------------------
```

```
            //-------------------------------------------------
            // Edge Detection Control
            //-------------------------------------------------
            case IDB_EDBUTTON:

                if (edgeDetectOn == 1)
                {
                    edgeDetectOn = 0;
                }

                else if (edgeDetectOn == 0)
                {
                    edgeDetectOn = 1;
                }

                break;

            //-------------------------------------------------
            // Exit and destroy application
            //-------------------------------------------------
            case IDB_EXITBUTTON:

                PostMessage(hwnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
                break;

    }
    //-------------------------------------------------
    // Scroll bar for pixel threshold
    //-------------------------------------------------
    case WM_HSCROLL:

        CurPos = GetScrollPos(hPixTrackBar, SB_CTL);

            switch (LOWORD(wParam))
            {

            case SB_THUMBPOSITION:
                CurPos = HIWORD(wParam);
                SetScrollPos(hPixTrackBar, SB_CTL, \
                CurPos, TRUE);
                PixelThreshold = PixelThreshold+\
                (CurPos-PixelThreshold);
            break;

            case SB_THUMBTRACK:
                CurPos = HIWORD(wParam);
                SetScrollPos(hPixTrackBar, SB_CTL, \
                CurPos, TRUE);
                PixelThreshold = PixelThreshold+\
                (CurPos-PixelThreshold);
            break;

            case SB_LINERIGHT:
                PixelThreshold++;
            break;

            case SB_LINELEFT:
                PixelThreshold--;
            break;

            case SB_PAGELEFT:
                PixelThreshold--;
            break;

            case SB_PAGERIGHT:
                PixelThreshold++;
            break;

            default:
            break;
        }
```

```
case WM_PAINT:
    //-----------------------------------------------------------------
    hdc = BeginPaint(hwnd, &ps);

    SetBkMode(hdc, TRANSPARENT);

    // Pixel Threshold Text.
    TextOut(hdc, 20, 270, "Pixel Threshold:", 16);
    TextOut(hdc, 20, 290, "0", 2);
    TextOut(hdc, 100, 290, "30", 2);

    // Camera Selection Text.
    TextOut(hdc, 140, 540, "Select Camera 1:", 16);
    TextOut(hdc, 140 + WEBCAM_WINDOW_X, 540, "Select Camera 2:", 16);
    TextOut(hdc, 140 + (2*WEBCAM_WINDOW_X), 540, "Select Camera 3:", 16);

    EndPaint(hwnd, &ps);
    //-----------------------------------------------------------------

    return 0;

case WM_TIMER:

    if( ! WebCamRunning )
    {
        return 0;
    }

    if( WebCamPaused )
    {
        return 0;
    }
    if(edgeDetectOn == 1)
    {
        if(CameraActive1 == 1)
        {
            if( StreamWebCamFrame1(ImageBuffer, ImageBufferLen, \
            &WebcamImageWidth, &WebcamImageHeight ) )
            {
                ProcessFrame1(ImageBuffer, ImageBuffer1, \
                WebcamImageWidth, \
                WebcamImageHeight);
            }
        }

        if(CameraActive2 == 1)
        {
            if( StreamWebCamFrame2(ImageBuffer, ImageBufferLen, \
            &WebcamImageWidth, &WebcamImageHeight ) )
            {
                ProcessFrame2(ImageBuffer, ImageBuffer1, \
                WebcamImageWidth, \
                WebcamImageHeight);
            }
        }

        if(CameraActive3 == 1)
        {
            if( StreamWebCamFrame3(ImageBuffer, ImageBufferLen, \
            &WebcamImageWidth, &WebcamImageHeight ) )
            {
                ProcessFrame3(ImageBuffer, ImageBuffer1, \
                WebcamImageWidth, \
                WebcamImageHeight);
            }
        }
    }

    return 0;
```

```
        case WM_CLOSE:
            ShowWindow(hwnd, SW_HIDE);
            return 0;

        case WM_DESTROY:

            //-------------------------------------------------
            KillTimer(hwnd, ID_TIMER);
            //-------------------------------------------------

            //-------------------------------------------------
            if( ImageBuffer )
            {
                free((void *)ImageBuffer);
            }
            //-------------------------------------------------

            //-------------------------------------------------
            if( ImageBuffer1 )
            {
                free((void *)ImageBuffer1);
            }
            //-------------------------------------------------
            CloseWebCamCapture();
            //-------------------------------------------------

            //-------------------------------------------------
            CoUninitialize();
            //-------------------------------------------------

            //-------------------------------------------------
            PostQuitMessage(0);
            //-------------------------------------------------

            return 0;

#if 0
        case WM_MOUSEMOVE:

            //-------------------------------------------------------------
            sprintf(tmpbuf, "Not in video");
            SetWindowText(hTextCursorPos, tmpbuf);
            //-------------------------------------------------------------

            return 0;
#endif

        case WM_KEYDOWN:
            //-------------------------------------------------------------
            nVirtKey = (int) wParam;    // virtual-key code
            //lKeyData = lParam;          // key data
            //-------------------------------------------------------------

            //-------------------------------------------------------------
            if( nVirtKey == VK_F8 )
            {
                //sprintf(tmpbuf, "key=%d", nVirtKey);
                //MessageBox(hwnd, tmpbuf, "debug", MB_OK);

                //-----------------------------------------
                if( ! WebCamRunning )
                {
                    // already running
                    return 0;
                }
                CloseWebCamCapture();

                WebCamRunning = 0;
                //-----------------------------------------
            }
```

```
            return 0;
    }
    return DefWindowProc (hwnd , message, wParam, lParam);
}

//=============================================================================
// Main window handle
//=============================================================================

HWND    CreateMainWindow(HINSTANCE hInstance, HINSTANCE hPrevInstance, \
        LPSTR lpszCmdLine, int nWinMode)
{

    // Define windows class
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(WNDCLASSEX);
    wndclass.style          = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW);
    wndclass.hbrBackground  = GetSysColorBrush(COLOR_3DFACE);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&wndclass))
    {
        MessageBox(NULL, TEXT ("This program requires Windows NT!"),\
        szAppName, MB_ICONERROR);
        return 0;
    }

    //-------------------------------------------------------------------------
    // Main Window
    //-------------------------------------------------------------------------

    hWnd = CreateWindowEx (WS_EX_CLIENTEDGE,
                           szAppName,                          // window class name
                           TEXT ("Webcam Capture Test"),       // window caption
                           WS_OVERLAPPEDWINDOW,                // window style
                           CW_USEDEFAULT,                      // initial x position
                           CW_USEDEFAULT,                      // initial y position
                           (3*WebcamImageWidth) + 180,         // initial x size
                           (2*WebcamImageHeight) + 180,        // initial y size
                           NULL,                               // parent window handle
                           NULL,                               // window menu handle
                           hInstance,                          // program instance handle
                           NULL);                              // creation parameters

    //-------------------------------------------------------------------------
    // Combo Boxes
    //-------------------------------------------------------------------------

    hWndComboBox1 = CreateWindowEx(0,"COMBOBOX","Camera 1",
                    CBS_DROPDOWNLIST | CBS_HASSTRINGS | WS_CHILD | WS_VISIBLE,
                    140, 560, 300, 160,
                    hWnd,(HMENU)IDC_COMBO1,hInstance,NULL);

    hWndComboBox2 = CreateWindowEx(0,"COMBOBOX","Camera 2",
                    CBS_DROPDOWNLIST | CBS_HASSTRINGS | WS_CHILD | WS_VISIBLE,
                    140 + WEBCAM_WINDOW_X, 560, 300, 160,
                    hWnd,(HMENU)IDC_COMBO2,hInstance,NULL);

    hWndComboBox3 = CreateWindowEx(0,"COMBOBOX","Camera 3",
                    CBS_DROPDOWNLIST | CBS_HASSTRINGS | WS_CHILD | WS_VISIBLE,
                    140 + (2*WEBCAM_WINDOW_X), 560, 300, 160,
                    hWnd,(HMENU)IDC_COMBO3,hInstance,NULL);
```

```
//-----------------------------------------------------------------------
// Buttons
//-----------------------------------------------------------------------

CreateWindowEx(BS_PUSHBUTTON,
               "button",
               "Start",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 30, 100, 20,
               hWnd, (HMENU)IDB_STARTBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Stop",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 55, 100, 20,
               hWnd, (HMENU)IDB_STOPBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Pause",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 80, 100, 20,
               hWnd, (HMENU)IDB_PAUSEBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Resume",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 105, 100, 20,
               hWnd, (HMENU)IDB_RESUMEBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Grab Left",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 150, 100, 20,
               hWnd, (HMENU)IDB_GRABLBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Grab Center",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 175, 100, 20,
               hWnd, (HMENU)IDB_GRABCBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Grab Right",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 200, 100, 20,
               hWnd, (HMENU)IDB_GRABRBUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
               "Grab All",
               WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
               20, 225, 100, 20,
               hWnd, (HMENU)IDB_GRABABUTTON,
               hInstance, (LPVOID)NULL);

CreateWindowEx(BS_PUSHBUTTON,
               "button",       // window class name
```

```
                    "Edge Detect",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 360, 100, 20,
                    hWnd, (HMENU)IDB_EDBUTTON,
                    hInstance, (LPVOID)NULL);

    CreateWindowEx(BS_PUSHBUTTON,
                    "button",       // window class name
                    "Exit",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 560, 100, 20,
                    hWnd, (HMENU)IDB_EXITBUTTON,
                    hInstance, (LPVOID)NULL);

    //------------------------------------------------------------------
    // Pixel Threshold Trackbar
    //------------------------------------------------------------------

    hPixTrackBar = CreateWindowEx(
                0,
                TRACKBAR_CLASS,
                "Trackbar Control",
                WS_CHILD | WS_VISIBLE | TBS_NOTICKS |
                TBS_HORZ | TBS_FIXEDLENGTH,
                20, 305,
                100, 30,
                hWnd,
                (HMENU)IDC_TRACK_PIX,
                hInstance,
                NULL
                );
                SendMessage(hPixTrackBar, TBM_SETRANGE,(WPARAM) TRUE,(LPARAM) \
                    MAKELONG(0, 30));
                SendMessage(hPixTrackBar, TBM_SETPOS, (WPARAM) TRUE,(LPARAM) 10);
                SendMessage(hPixTrackBar, TBM_SETPAGESIZE, 0, (LPARAM) 1);
                //SetFocus(hPixTrackBar);

    //Show Window
    ShowWindow(hWnd, nWinMode);
    UpdateWindow(hWnd);


    return hWnd;
}


//=================================================================
// Create Capture Windows
//=================================================================

LRESULT CALLBACK WndProcCaptureWindow(HWND hWnd, UINT message, WPARAM wParam,\
        LPARAM lParam)
{
    PAINTSTRUCT     ps;

    switch( message )
    {
        case WM_CREATE:
            return 0;

        case WM_PAINT:
            BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            return 0;

        case WM_CLOSE:
            PostMessage(hWnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
```

```
            return 0;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
}

//=============================================================
// Capture window handle
//=============================================================
HWND    CreateCaptureWindow(HWND hParentWnd)
{
    //HWND          hWnd;
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(WNDCLASSEX);
    wndclass.style          = 0;
    wndclass.lpfnWndProc    = WndProcCaptureWindow;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = NULL;
    wndclass.hCursor        = NULL;
    wndclass.hbrBackground  = GetSysColorBrush(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.hIconSm        = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.lpszClassName  = "CaptureWindowClass";

    if( ! RegisterClassEx(&wndclass) )
        return 0;

    hVidWnd1 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "CaptureWindowClass", "WebCam_Window",
                              WS_CHILD | WS_VISIBLE,
                              140, 20,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              hParentWnd,
                              NULL,
                              hInstance,
                              NULL);

    hVidWnd2 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "CaptureWindowClass", "WebCam_Window",
                              WS_CHILD | WS_VISIBLE,
                              140 + WEBCAM_WINDOW_X, 20,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              hParentWnd,
                              NULL,
                              hInstance,
                              NULL);

     hVidWnd3 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "CaptureWindowClass", "WebCam_Window",
                              WS_CHILD | WS_VISIBLE,
                              140 + (2*WEBCAM_WINDOW_X), 20,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              hParentWnd,
                              NULL,
                              hInstance,
                              NULL);


    ShowWindow(hWnd, SW_SHOW);
    UpdateWindow(hWnd);

    return hWnd;
}

//=============================================================
// Create Frame Grab Windows
//=============================================================
```

```
LRESULT CALLBACK WndProcGrabWindow(HWND hWnd, UINT message, WPARAM wParam,\
        LPARAM lParam)
{
    PAINTSTRUCT      ps;

    switch( message )
    {
        case WM_CREATE:
            return 0;

        case WM_PAINT:
            BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            return 0;

        case WM_CLOSE:
            PostMessage(hWnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
}

//=============================================================
// Grab window handle
//=============================================================
HWND    CreateGrabWindow(HWND hParentWnd)
{
    //HWND        hWnd;
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(WNDCLASSEX);
    wndclass.style          = 0;
    wndclass.lpfnWndProc    = WndProcCaptureWindow;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = NULL;
    wndclass.hCursor        = NULL;
    wndclass.hbrBackground  = GetSysColorBrush(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.hIconSm        = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.lpszClassName  = "GrabWindowClass";

    if( ! RegisterClassEx(&wndclass) )
        return 0;

    hGraWnd1 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "GrabWindowClass", "WebCam_Window",
                              WS_CHILD | WS_VISIBLE,
                              140, 40 + WEBCAM_WINDOW_Y,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              hParentWnd,
                              NULL,
                              hInstance,
                              NULL);

    hGraWnd2 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "GrabWindowClass", "WebCam_Window",
                              WS_CHILD | WS_VISIBLE,
                              140 + WEBCAM_WINDOW_X, 40 + WEBCAM_WINDOW_Y,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              hParentWnd,
                              NULL,
                              hInstance,
```

```
                                NULL);

    hGraWnd3 = CreateWindowEx(WS_EX_CLIENTEDGE,
                                "GrabWindowClass", "WebCam_Window",
                                WS_CHILD | WS_VISIBLE,
                                140 + (2*WEBCAM_WINDOW_X), 40 + WEBCAM_WINDOW_Y,
                                WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                                hParentWnd,
                                NULL,
                                hInstance,
                                NULL);

    ShowWindow(hWnd, SW_SHOW);
    UpdateWindow(hWnd);

    if(hWnd)
    {
        HRESULT hr;
    }

    return hWnd;
}
```

C.2 Source - calldll.c

Image capure DLL source code:

```
//------------------------------------------------------------------------------
//
//  dshow_webcam.c - dshow_webcam.dll
//  Library Access for Matlab.
//  Created by Adam Cox
//  Based on code provided by John Leis - USQ
//
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
#define WIN32_LEAN_AND_MEAN
#define STRICT

#pragma comment(lib,"user32.lib")
#pragma comment(lib,"gdi32.lib")
#pragma comment(lib,"shell32.lib")
#pragma comment(lib,"strmiids.lib")
#pragma comment(lib,"ole32.lib")
#pragma comment(lib,"amstrmid.lib")
#pragma comment(lib,"oleaut32.lib")
#pragma comment(lib,"uuid.lib")
#pragma comment(lib,"quartz.lib")

#include <windows.h>
#include <stdio.h>
#include <dshow.h> // Link with strmiids.lib and quartz.lib
#include <stddef.h>
#include <shlobj.h>
#include <string.h>
#include <ocidl.h>
#include "dshow_webcam.h"
//------------------------------------------------------------------------------

//==============================================================================
// Global Definitions
//==============================================================================
IBaseFilter *pSrc1 = NULL;
IBaseFilter *pSrc2 = NULL;
IBaseFilter *pEnumSrc1 = NULL;
IBaseFilter *pEnumSrc2 = NULL;
IEnumPins *pEnum1 = NULL;
IEnumPins *pEnum2 = NULL;
IGraphBuilder *pGraphBuilder1 = NULL;
IGraphBuilder *pGraphBuilder2 = NULL;
IMediaControl *pMediaControl1 = NULL;
IMediaControl *pMediaControl2 = NULL;
IMediaEventEx *pMediaEvent1 = NULL;
IMediaEventEx *pMediaEvent2 = NULL;
ICaptureGraphBuilder2 *pCaptureGraphBuilder1 = NULL;
ICaptureGraphBuilder2 *pCaptureGraphBuilder2 = NULL;
IVMRWindowlessControl *VMRpVidWin1 = NULL;
IVMRWindowlessControl *VMRpVidWin2 = NULL;
IPin *m_pCamOutPin1 = NULL;
IPin *m_pCamOutPin2 = NULL;
ISpecifyPropertyPages *pSpecPropPage1 = NULL;
ISpecifyPropertyPages *pSpecPropPage2 = NULL;

//------------------------------------------------------------------------------
AM_MEDIA_TYPE amt;

//------------------------------------------------------------------------------
//Bitmap definitions for header info.
```

```
typedef LPBITMAPINFOHEADER PDIB;
// Constants
#define BFT_BITMAP 0x4d42   /* 'BM' */
// Macros
#define DibNumColors(lpbi) ((lpbi)->biClrUsed == 0 && (lpbi)->biBitCount <= 8\
                           ? (int)(1 << (int)(lpbi)->biBitCount)\
                           : (int)(lpbi)->biClrUsed)

#define DibSize(lpbi)      ((lpbi)->biSize + (lpbi)->biSizeImage +\
                            (int)(lpbi)->biClrUsed * sizeof(RGBQUAD))
#define DibPaletteSize(lpbi)    (DibNumColors(lpbi) * sizeof(RGBQUAD))

// bmp filename counters
int leftFileCount = 0;
int centerFileCount = 0;
int rightFileCount = 0;

//---------------------------------------------------------------
#define WEBCAM_WINDOW_X      320
#define WEBCAM_WINDOW_Y      240
//---------------------------------------------------------------
int cameraNum = 0; //Number of current enumerated capture devices.

int CurPos; // Trackbar Position
int DevMenuIndex = 0; // Index for menu order.

int CameraActive1 = 0;
int CameraActive2 = 0;

int ItemIndex1 = 0;

int WebCamRunning = 0;
int WebCamPaused = 0;
int WebCamInitialized = 0;

#define DEFAULT_CANVAS_WIDTH     320
#define DEFAULT_CANVAS_HEIGHT    240

static int WebcamImageWidth = DEFAULT_CANVAS_WIDTH;
static int WebcamImageHeight = DEFAULT_CANVAS_HEIGHT;

int *pWidth;
int *pHeight;

*pWidth = &WebcamImageWidth;
*pHeight = &WebcamImageHeight;


unsigned int LeftFileCount = 1;
unsigned int RightFileCount = 1;

//=================================================================
// Define Windows
//=================================================================
HINSTANCE    hInstance;

HWND     CreateCaptureWindow1(HWND hParentWnd);
HWND     CreateCaptureWindow2(HWND hParentWnd);
HWND     hWnd;
HWND     hMainWnd; // Main Window handle.
HWND     hVidWnd1; // Camera Windows
HWND     hVidWnd2;
LRESULT CALLBACK WndProcCaptureWindow1(HWND hwnd, UINT message, \
         WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndProcCaptureWindow2(HWND hwnd, UINT message, \
         WPARAM wParam, LPARAM lParam);
//=================================================================

//=================================================================
// Function Prototypes
//=================================================================
```

```
int      FindCaptureDevice(void);
//-------------------------------------------------------------------------
int      WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void *p);
int      WINAPI mydll_init(HANDLE h, DWORD reason, void *foo);
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// *** make sure these are declared in the header for MATLAB  ***
int      InitWebCamCapture1(void);
int      InitWebCamCapture2(void);

int      WebCam1(void);
int      WebCam2(void);

int      InitVideoWindow1(HWND hVidWnd1, int *pWidth, int *pHeight);
int      InitVideoWindow2(HWND hVidWnd2, int *pWidth, int *pHeight);

int      InitializeWindowlessVMR1(void);
int      InitializeWindowlessVMR2(void);

int      ReleaseFilters(void);

int      StreamWebCamFrame1(unsigned char *pFrameOut, unsigned long FrameBufferLen,\
         int *pWidth, int *pHeight);
int      StreamWebCamFrame2(unsigned char *pFrameOut, unsigned long FrameBufferLen,\
         int *pWidth, int *pHeight);

int      GrabWebCamFrame1(void);
int      GrabWebCamFrame2(void);
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void *p)
{
    return 1;
}

int WINAPI mydll_init(HANDLE h, DWORD reason, void *foo)
{
    return 1;
}
//-------------------------------------------------------------------------

//=========================================================================
// Find Capture Devices and Create Filters
//=========================================================================

int FindCaptureDevice(void)
{

    HRESULT hr = S_OK;
    cameraNum = 0;

    IMoniker *pMoniker= NULL;
    ICreateDevEnum *pDevEnum= NULL;
    IEnumMoniker *pClassEnum= NULL;
    IPropertyBag *pPropBag= NULL;

    // Create the system device enumerator
    hr = CoCreateInstance (&CLSID_SystemDeviceEnum,
                NULL,
                CLSCTX_INPROC,
                &IID_ICreateDevEnum,
                (void **) &pDevEnum);

    if (FAILED(hr))
    {
        MessageBox(0, TEXT("Device Enumeration Failed!"), 0, 0);
        return hr;
    }
```

```
// Create an enumerator for the video capture devices
if (SUCCEEDED(hr))
{
    // Create an enumerator for the category.
    hr = pDevEnum->CreateClassEnumerator (&CLSID_VideoInputDeviceCategory, \
        &pClassEnum, 0);
    if (hr == S_FALSE)
    {
        hr = VFW_E_NOT_FOUND;  // The category is empty. Treat as an error.
    }
    pDevEnum->Release();

    if (FAILED(hr))
    {
        MessageBox(0, TEXT("No Devices Detected!"), 0, 0);
        return hr;
    }
}

if (SUCCEEDED(hr))
{
    // If there are no enumerators for the requested type, then
    // CreateClassEnumerator will succeed, but pClassEnum will be NULL.
    if (pClassEnum == NULL)
    {
        MessageBox(0, TEXT("Failed"), 0, 0);
        hr = E_FAIL;
        return hr;
    }
}

while (pClassEnum->Next(1, &pMoniker, NULL) == S_OK)
{

    HRESULT hr = pMoniker->BindToStorage(0, 0, &IID_IPropertyBag, \
        (void **)&pPropBag);
    if (FAILED(hr))
    {
        MessageBox(0, TEXT("Binding to Storage Failed"), 0, 0);
        pMoniker->Release();
        continue;
    }

    VARIANT var;
    var.vt = VT_BSTR;

    // Get description or friendly name.
    hr = pPropBag->Read(L"Description", &var, 0);
    if (FAILED(hr))
    {
        hr = pPropBag->Read(L"FriendlyName", &var, 0);

        char szName[256];
        // Convert BSTR
        WideCharToMultiByte(CP_ACP,0,var.bstrVal,-1,szName,256,0,0);
        MessageBox(0, szName, TEXT("Camera Found: "), 0);

    }
    if (SUCCEEDED(hr))
    {
        VariantClear(&var);
        SysFreeString(var.bstrVal);
    }

    hr = pPropBag->Write(L"FriendlyName", &var);

    switch(cameraNum)
        {
        case 0:
            // Bind Moniker to a filter object
            pMoniker->BindToObject(0,0,&IID_IBaseFilter, (void**)&pSrc1);
```

```
                cameraNum ++;
                CameraActive1 = 1;
                break;

            case 1:
                // Bind Moniker to a filter object
                pMoniker->BindToObject(0,0,&IID_IBaseFilter, (void**)&pSrc2);
                cameraNum ++;
                CameraActive2 = 1;
                break;


            default:
                return hr;
            }


        pPropBag->Release();
        pMoniker->Release();

    }
    pClassEnum->Release();

    return 1;
}

//============================================================
// Create Capture Device #1 Filter
//============================================================
int     InitWebCamCapture1(void)
{

    HRESULT hr;

    // Create the filter graph manager and query for interfaces.
    CoCreateInstance(&CLSID_FilterGraph, //Class ID for COM object
                     NULL,
                     CLSCTX_INPROC_SERVER,
                     &IID_IGraphBuilder, // Interface ID
                     (void **)&pGraphBuilder1); // Pointer back to FGM

    // Media Control provides methods for flow of data through the filter
    // graph i.e. play, stop, pause...
    pGraphBuilder1->QueryInterface(&IID_IMediaControl, // Interface ID
                     (void **)&pMediaControl1); // Pointer back to MC

    // Media Event provides methods for retrieving event notifications.
    pGraphBuilder1->QueryInterface(&IID_IMediaEventEx, // Interface ID
                     (void **)&pMediaEvent1); // Pointer back to ME

    // Capture Graph Builder captures live video.
    CoCreateInstance(&CLSID_CaptureGraphBuilder2, //Class ID for COM object
                     NULL,
                     CLSCTX_INPROC,
                     &IID_ICaptureGraphBuilder2, // Interface ID
                     (void **)&pCaptureGraphBuilder1); // Pointer back to CGB2

    // Set the filter graph to capture graph.
    pCaptureGraphBuilder1->SetFiltergraph(pGraphBuilder1);

    // Attach the filter graph to capture graph.
    pGraphBuilder1->AddFilter(pSrc1, L"Video_Capture");

    // Enumerate pins from Capture filter.
    pSrc1->EnumPins(&pEnum1);
    pEnum1->Reset();
    pEnum1->Next(1, &m_pCamOutPin1, NULL);

    // Pin Properties.
    hr = m_pCamOutPin1->QueryInterface(&IID_ISpecifyPropertyPages, \
        (void **)&pSpecPropPage1);
```

```
    if (SUCCEEDED(hr))
    {
        PIN_INFO PinInfo;
        m_pCamOutPin1->QueryPinInfo(&PinInfo);

        // Show the page.
        CAUUID caGUID;
        pSpecPropPage1->GetPages(&caGUID);
        OleCreatePropertyFrame(
            hMainWnd,
            0,
            0,
            L"Capture Device 1",
            1,
            (IUnknown **)&(m_pCamOutPin1),
            caGUID.cElems,
            caGUID.pElems,
            0,
            0,
        NULL);
        CoTaskMemFree(caGUID.pElems);
        PinInfo.pFilter->Release();
    }

    InitializeWindowlessVMR1();

    if( ! InitVideoWindow1(hVidWnd1, pWidth, pHeight) )
    {
        return 0;
    }


    hr = pGraphBuilder1->Render(m_pCamOutPin1);
    if (FAILED(hr))
    {
        return 0;
    }

    // Run live capture from device.
    pMediaControl1->Run();

    pEnum1->Release();

    WebCamInitialized = 1;
    WebCamPaused = 0;


    return 1;
}

//=============================================================
// Create Capture Device #2 Filter
//=============================================================
int     InitWebCamCapture2(void)
{

    HRESULT hr;

    // Create the filter graph manager and query for interfaces.
    CoCreateInstance(&CLSID_FilterGraph, //Class ID for COM object
                     NULL,
                     CLSCTX_INPROC_SERVER,
                     &IID_IGraphBuilder, // Interface ID
                     (void **)&pGraphBuilder2); // Pointer back to FGM

    // Media Control provides methods for flow of data through the filter
    // graph i.e. play, stop, pause...
    pGraphBuilder2->QueryInterface(&IID_IMediaControl, // Interface ID
                     (void **)&pMediaControl2); // Pointer back to MC

    // Media Event provides methods for retrieving event notifications.
```

```
pGraphBuilder2->QueryInterface(&IID_IMediaEventEx, // Interface ID
                (void **)&pMediaEvent2); // Pointer back to ME

// Capture Graph Builder captures live video.
CoCreateInstance(&CLSID_CaptureGraphBuilder2, //Class ID for COM object
                NULL,
                CLSCTX_INPROC,
                &IID_ICaptureGraphBuilder2, // Interface ID
                (void **)&pCaptureGraphBuilder2); // Pointer back to CGB2

// Set the filter graph to capture graph.
pCaptureGraphBuilder2->SetFiltergraph(pGraphBuilder2);

// Attach the filter graph to capture graph.
pGraphBuilder2->AddFilter(pSrc2, L"Video Capture");

// Enumerate pins from Capture filter.
pSrc2->EnumPins(&pEnum2);
pEnum2->Reset();
pEnum2->Next(1, &m_pCamOutPin2, NULL);

// Pin Properties.
hr = m_pCamOutPin2->QueryInterface(&IID_ISpecifyPropertyPages, \
    (void **)&pSpecPropPage2);
if (SUCCEEDED(hr))
{
    PIN_INFO PinInfo;
    m_pCamOutPin2->QueryPinInfo(&PinInfo);

    // Show the page.
    CAUUID caGUID;
    pSpecPropPage2->GetPages(&caGUID);
    OleCreatePropertyFrame(
        hMainWnd,
        0,
        0,
        L"Capture Device 2",
        1,
        (IUnknown **)&(m_pCamOutPin2),
        caGUID.cElems,
        caGUID.pElems,
        0,
        0,
    NULL);
    CoTaskMemFree(caGUID.pElems);
    PinInfo.pFilter->Release();
}

InitializeWindowlessVMR2();

if( ! InitVideoWindow2(hVidWnd2, pWidth, pHeight) )
{
    return 0;
}


hr = pGraphBuilder2->Render(m_pCamOutPin2);
if (FAILED(hr))
{
    return 0;
}

// Run live capture from device.
pMediaControl2->Run();

pEnum2->Release();

WebCamInitialized = 1;
WebCamPaused = 0;
```

```
    return 1;
}

//=============================================================================
// Video Window #1 Function
//=============================================================================

// Collect video window dimensions
int      InitVideoWindow1(HWND hVidWnd1, int *pWidth, int *pHeight)
{
    HRESULT hr;
    RECT rcDest;
    IAMStreamConfig  *pStreampConfig = NULL;
    IEnumMediaTypes *pEnumMediaType = NULL;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;
    VIDEOINFOHEADER *vidInfoHead = NULL;

    hr = m_pCamOutPin1->EnumMediaTypes(&pEnumMediaType);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    while(pEnumMediaType->Next(1, &pmt, 0) == S_OK)
    {
        if( memcmp((void *)& pmt->formattype, (void *)&FORMAT_VideoInfo, \
            sizeof(GUID)) == 0 )
        {

            vidInfoHead = (VIDEOINFOHEADER *)pmt->pbFormat;
            {
                pfnt = pmt;
                break;
            }
        }
    }

    pEnumMediaType->Release();

    hr = m_pCamOutPin1->QueryInterface(&IID_IAMStreamConfig, \
        (void **)&pStreampConfig);
    if(FAILED(hr))
    {
        return 0;
    }

    if( ! pfnt )
    {
        return 0;
    }

    hr = pStreampConfig->SetFormat(pfnt);

    CoTaskMemFree((void *)pfnt);

    hr = pStreampConfig->GetFormat(&pfnt);
    if(FAILED(hr))
    {
        return 0;
    }

    // Collect image width and height from video header.
    *pWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biWidth;
    *pHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biHeight;

    CoTaskMemFree((void *)pfnt);

    // Force preview to fixed size 320*240
    rcDest.left = 0;
    rcDest.top = 0;
    rcDest.right = WEBCAM_WINDOW_X;
```

```
    rcDest.bottom = WEBCAM_WINDOW_Y;
    hr = VMRpVidWin1->SetVideoPosition(NULL, &rcDest);

    pStreampConfig->Release();

    return 1;
}


//=============================================================
// Video Window #2 Function
//=============================================================

// Collect video window dimensions
int     InitVideoWindow2(HWND hVidWnd2, int *pWidth, int *pHeight)
{
    HRESULT hr;
    RECT rcDest;
    IAMStreamConfig  *pStreampConfig = NULL;
    IEnumMediaTypes *pEnumMediaType = NULL;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;
    VIDEOINFOHEADER *vidInfoHead = NULL;

    hr = m_pCamOutPin2->EnumMediaTypes(&pEnumMediaType);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    while(pEnumMediaType->Next(1, &pmt, 0) == S_OK)
    {
        if( memcmp((void *)& pmt->formattype, (void *)&FORMAT_VideoInfo, \
            sizeof(GUID)) == 0 )
        {

            vidInfoHead = (VIDEOINFOHEADER *)pmt->pbFormat;
            {
                pfnt = pmt;
                break;
            }
        }
    }

    pEnumMediaType->Release();

    hr = m_pCamOutPin2->QueryInterface(&IID_IAMStreamConfig, \
        (void **)&pStreampConfig);
    if(FAILED(hr))
    {
        return 0;
    }

    if( ! pfnt )
    {
        return 0;
    }

    hr = pStreampConfig->SetFormat(pfnt);

    CoTaskMemFree((void *)pfnt);

    hr = pStreampConfig->GetFormat(&pfnt);
    if(FAILED(hr))
    {
        return 0;
    }

    // Collect image width and height from video header.
    *pWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biWidth;
    *pHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biHeight;
```

```
    CoTaskMemFree((void *)pfnt);

    // Force preview to fixed size 320*240
    rcDest.left = 0;
    rcDest.top = 0;
    rcDest.right = WEBCAM_WINDOW_X;
    rcDest.bottom = WEBCAM_WINDOW_Y;
    hr = VMRpVidWin2->SetVideoPosition(NULL, &rcDest);

    pStreampConfig->Release();

    return 1;
}


//=============================================================
// Windoless VMR #1 Function
//=============================================================

// Windowless video control for video window placement.
int     InitializeWindowlessVMR1(void)
{

    IBaseFilter       *pVmr1 = NULL;

    IVMRFilterConfig *pConfig1 = NULL;


    HRESULT hr;


    // Create the VMR and add it to the filter graph.
    hr = CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
                          CLSCTX_INPROC, &IID_IBaseFilter, (void**)&pVmr1);

    if(FAILED(hr))
    {
        return 0;
    }

    hr = pGraphBuilder1->AddFilter(pVmr1, L"Video Mixing Renderer");
    if(FAILED(hr))
    {
        return 0;
    }

    // Set the rendering mode and number of streams.

    hr = pVmr1->QueryInterface(&IID_IVMRFilterConfig, (void**)&pConfig1);
    if(FAILED(hr))
    {
        return 0;
    }



    pConfig1->SetRenderingMode(VMRMode_Windowless);
    pConfig1->Release();

    hr = pVmr1->QueryInterface(&IID_IVMRWindowlessControl, \
        (void**)&VMRpVidWin1);
    if(FAILED(hr))
    {
        return 0;
    }


    //Set VMR windowless output to windows handle
    VMRpVidWin1->SetVideoClippingWindow(hVidWnd1);

    //Release VMR control
```

```
    pVmr1->Release();

        return 1;
}


//=====================================================================
// Windoless VMR #2 Function
//=====================================================================

// Windowless video control for video window placement.
int     InitializeWindowlessVMR2(void)
{

    IBaseFilter        *pVmr2 = NULL;

    IVMRFilterConfig *pConfig2 = NULL;


    HRESULT hr;


    // Create the VMR and add it to the filter graph.
    hr = CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
                          CLSCTX_INPROC, &IID_IBaseFilter, (void**)&pVmr2);

    if(FAILED(hr))
    {
        return 0;
    }

    hr = pGraphBuilder2->AddFilter(pVmr2, L"Video Mixing Renderer");
    if(FAILED(hr))
    {
        return 0;
    }

    // Set the rendering mode and number of streams.

    hr = pVmr2->QueryInterface(&IID_IVMRFilterConfig, (void**)&pConfig2);
    if(FAILED(hr))
    {
        return 0;
    }



    pConfig2->SetRenderingMode(VMRMode_Windowless);
    pConfig2->Release();

    hr = pVmr2->QueryInterface(&IID_IVMRWindowlessControl, \
        (void**)&VMRpVidWin2);
    if(FAILED(hr))
    {
        return 0;
    }


    //Set VMR windowless output to windows handle
    VMRpVidWin2->SetVideoClippingWindow(hVidWnd2);

    //Release VMR control
    pVmr2->Release();

        return 1;
}

//=====================================================================
// Grab Frame From Webcam 1
//=====================================================================
```

```
// Grab frame and save as bitmap format

int GrabWebCamFrame1(void)
{

    HRESULT hr;
    BYTE* lpCurrImage = NULL;

    if (CameraActive1==1)
    {
        if(SUCCEEDED(hr = VMRpVidWin1->GetCurrentImage(&lpCurrImage)))
        {

            BITMAPFILEHEADER    hdr;
            DWORD               dwSize, dwWritten;
            LPBITMAPINFOHEADER  pdib = (LPBITMAPINFOHEADER) lpCurrImage;

            const char *name = "left";
            char leftFileName[32] = {0};

            sprintf(leftFileName, "%s%d.bmp", name, leftFileCount);
            leftFileCount++;

            // Create handle and output bmp file
            HANDLE WebcamFrame1 = CreateFile(leftFileName, GENERIC_WRITE,
                                  FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                                  FILE_ATTRIBUTE_NORMAL, 0);

            if (WebcamFrame1 == INVALID_HANDLE_VALUE)
                return 0;

            // Initialize the bitmap header
            dwSize = DibSize(pdib);
            hdr.bfType          = BFT_BITMAP;
            hdr.bfSize          = dwSize + sizeof(BITMAPFILEHEADER);
            hdr.bfReserved1     = 0;
            hdr.bfReserved2     = 0;
            hdr.bfOffBits       = (DWORD)sizeof(BITMAPFILEHEADER) + \
                                  pdib->biSize + DibPaletteSize(pdib);

            // Write the bitmap to the file
            WriteFile(WebcamFrame1, (LPCVOID) &hdr, sizeof(BITMAPFILEHEADER),\
            &dwWritten, 0);
            WriteFile(WebcamFrame1, (LPCVOID) pdib, dwSize, &dwWritten, 0);

            // Close the file
            CloseHandle(WebcamFrame1);

            // Free the image data returned from GetCurrentImage()
            CoTaskMemFree(lpCurrImage);
        }
        else
        {
            MessageBox(0, TEXT("Frame grab failed on Device 1!"), 0, 0);
            return 0;
        }
    }
    else
    {
        MessageBox(0, TEXT("Cannot Find Device 1!"), 0, 0);
        return 0;
    }
return 1;
}


//=====================================================================
// Grab Frame From Webcam 2
//=====================================================================

// Grab frame and save as bitmap format
```

```
int GrabWebCamFrame2(void)
{
    HRESULT hr;
    BYTE* lpCurrImage = NULL;

    if (CameraActive2==1)
    {
        if(SUCCEEDED(hr = VMRpVidWin2->GetCurrentImage(&lpCurrImage)))
        {

            BITMAPFILEHEADER    hdr;
            DWORD               dwSize, dwWritten;
            LPBITMAPINFOHEADER  pdib = (LPBITMAPINFOHEADER) lpCurrImage;

            const char *name = "right";
            char rightFileName[32] = {0};

            sprintf(rightFileName, "%s%d.bmp", name, rightFileCount);
            rightFileCount++;

            // Create handle and output bmp file
            HANDLE WebcamFrame2 = CreateFile(rightFileName, GENERIC_WRITE,
                                             FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                                             FILE_ATTRIBUTE_NORMAL, 0);

            if (WebcamFrame2 == INVALID_HANDLE_VALUE)
                return 0;

            // Initialize the bitmap header
            dwSize = DibSize(pdib);
            hdr.bfType          = BFT_BITMAP;
            hdr.bfSize          = dwSize + sizeof(BITMAPFILEHEADER);
            hdr.bfReserved1     = 0;
            hdr.bfReserved2     = 0;
            hdr.bfOffBits       = (DWORD)sizeof(BITMAPFILEHEADER) + \
            pdib->biSize + DibPaletteSize(pdib);

            // Write the bitmap to the file
            WriteFile(WebcamFrame2, (LPCVOID) &hdr, sizeof(BITMAPFILEHEADER),\
            &dwWritten, 0);
            WriteFile(WebcamFrame2, (LPCVOID) pdib, dwSize, &dwWritten, 0);

            // Close the file
            CloseHandle(WebcamFrame2);

            // Free the image data returned from GetCurrentImage()
            CoTaskMemFree(lpCurrImage);
        }
        else
        {
            MessageBox(0, TEXT("Frame grab failed on Device 2!"), 0, 0);
            return 0;
        }
    }
    else
    {
        MessageBox(0, TEXT("Cannot Find Device 2!"), 0, 0);
        return 0;
    }
return 1;
}



//=========================================================================
// Webcam 1 Window
//=========================================================================

int WebCam1(void)
{
```

```
    if (CameraActive1==1)
    {
        hWnd = CreateCaptureWindow1(hMainWnd);

        // Start camera
        InitWebCamCapture1();
    }
    else
    {
        MessageBox(0, TEXT("Cannot Find Device 1!"), 0, 0);
        return 0;
    }
    return 1;
}

//=============================================================
// Webcam 2 Window
//=============================================================

int WebCam2(void)
{
    if (CameraActive2==1)
    {
        hWnd = CreateCaptureWindow2(hMainWnd);

        // Start camera
        InitWebCamCapture2();
    }
    else
    {
        MessageBox(0, TEXT("Cannot Find Device 2!"), 0, 0);
        return 0;
    }
    return 1;
}

//=============================================================
// Create Capture Window 1
//=============================================================

LRESULT CALLBACK WndProcCaptureWindow1(HWND hWnd, UINT message, \
        WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT      ps;

    switch( message )
    {
        case WM_CREATE:
            return 0;

        case WM_PAINT:
            BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            return 0;

        case WM_CLOSE:
            ReleaseFilters();
            DestroyWindow(hWnd);
            PostMessage(hWnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            ReleaseFilters();
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
}
```

```
//=============================================================
// Capture window 1 handle
//=============================================================
HWND     CreateCaptureWindow1(HWND hParentWnd)
{
    //HWND          hWnd;
    WNDCLASSEX   wndclass;

    wndclass.cbSize          = sizeof(WNDCLASSEX);
    wndclass.style           = 0;
    wndclass.lpfnWndProc     = WndProcCaptureWindow1;
    wndclass.cbClsExtra      = 0;
    wndclass.cbWndExtra      = 0;
    wndclass.hInstance       = hInstance ;
    wndclass.hIcon           = NULL;
    wndclass.hCursor         = NULL;
    wndclass.hbrBackground   = GetSysColorBrush(WHITE_BRUSH);
    wndclass.lpszMenuName    = NULL;
    wndclass.hIconSm         = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.lpszClassName   = "CaptureWindowClass1";

    if( ! RegisterClassEx(&wndclass) )
        return 0;

    hVidWnd1 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "CaptureWindowClass1", "WebCam Window",
                              WS_VISIBLE | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX,
                              0, 0,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              NULL,
                              NULL,
                              hInstance,
                              NULL);



    ShowWindow(hWnd, SW_SHOW);
    UpdateWindow(hWnd);

    return hWnd;
}

//=============================================================
// Create Capture Window 2
//=============================================================

LRESULT CALLBACK WndProcCaptureWindow2(HWND hWnd, UINT message,\
        WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT      ps;

    switch( message )
    {
        case WM_CREATE:
            return 0;

        case WM_PAINT:
            BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            return 0;

        case WM_CLOSE:
            ReleaseFilters();
            DestroyWindow(hWnd);
            PostMessage(hWnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            ReleaseFilters();
            PostQuitMessage(0);
```

```
            return 0;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
}

//=============================================================
// Capture window 2 handle
//=============================================================
HWND    CreateCaptureWindow2(HWND hParentWnd)
{
    //HWND        hWnd;
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(WNDCLASSEX);
    wndclass.style          = 0;
    wndclass.lpfnWndProc    = WndProcCaptureWindow2;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance ;
    wndclass.hIcon          = NULL;
    wndclass.hCursor        = NULL;
    wndclass.hbrBackground  = GetSysColorBrush(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.hIconSm        = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.lpszClassName  = "CaptureWindowClass2";

    if( ! RegisterClassEx(&wndclass) )
        return 0;

    hVidWnd2 = CreateWindowEx(WS_EX_CLIENTEDGE,
                              "CaptureWindowClass2", "WebCam_Window",
                              WS_VISIBLE | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX,
                              0, 0,
                              WEBCAM_WINDOW_X, WEBCAM_WINDOW_Y,
                              NULL,
                              NULL,
                              hInstance,
                              NULL);



    ShowWindow(hWnd, SW_SHOW);
    UpdateWindow(hWnd);

    return hWnd;
}


//=============================================================
// Release Filters
//=============================================================

int ReleaseFilters(void)
{

// Release all filters.
if (cameraNum==1)
{
    PostQuitMessage(0);
    pMediaControl1->Stop();
    m_pCamOutPin1->Disconnect();
    m_pCamOutPin1->Release();
    pSrc1->Release();
    pMediaControl1->Release();
    pMediaEvent1->Release();
}

if (cameraNum==2)
{
```

```
    PostQuitMessage(0);
    pMediaControl1->Stop();
    m_pCamOutPin1->Disconnect();
    m_pCamOutPin1->Release();
    pSrc1->Release();
    pMediaControl1->Release();
    pMediaEvent1->Release();

    pMediaControl2->Stop();
    m_pCamOutPin2->Disconnect();
    m_pCamOutPin2->Release();
    pSrc2->Release();
    pMediaControl2->Release();
    pMediaEvent2->Release();
}

return 1;
}

//-------------------------------------------------------------------------
#if 0
// empty
//int main()
int main(int argc, char *argv)
{
    return 1;
}
#endif
```

C.3 Source - directshow_webcam.c

C test source code for DLL:

```
/* calldll.c
 *
 * Basic Test Application for directshow_webcam.c
 * Developed by Adam Cox
 * Based on code provided by John Leis - USQ
 */
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>

//-------------------------------------------------------------------------
// for static call
int      FindCaptureDevice(void);
int      WebCam1(void);
int      WebCam2(void);
int      GrabWebCamFrame1(void);
int      GrabWebCamFrame2(void);
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// for dynamic call
typedef void (*DLLPROC)(long x, long *py, long *parr, char *strarg);

// module handle for module containing call
static HMODULE  hDllMod = (HMODULE)NULL;

// ptr to actual proc
static DLLPROC  DllProc = (DLLPROC)NULL;

void     LoadDLL(void);
void     UnloadDLL(void);

//-------------------------------------------------------------------------
// Global variables

// The main window class name.
static TCHAR szWindowClass[] = _T("win32app");

// The string that appears in the application's title bar.
static TCHAR szTitle[] = _T("DLL Test App");

HINSTANCE hInst;

// Forward declarations of functions included in this code module:
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{

    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
```

```
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = NULL;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));

    if (!RegisterClassEx(&wcex))
    {


        return 1;
    }

    hInst = hInstance; // Store instance handle in our global variable

    HWND hWnd = CreateWindow(
        szWindowClass,
        szTitle,
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        500, 100,
        NULL,
        NULL,
        hInstance,
        NULL
    );

    if (!hWnd)
    {
        return 1;
    }

    ShowWindow(hWnd,nCmdShow);
    UpdateWindow(hWnd);

    LoadDLL();

    FindCaptureDevice();

    WebCam1();
    WebCam2();
    GrabWebCamFrame1();
    GrabWebCamFrame2();
    UnloadDLL();

    // Main message loop:
    MSG msg;

    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int) msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    TCHAR greeting[] = _T("DLL Test!");

    switch (message)
    {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);

        TextOut(hdc,
            5, 5,
            greeting, _tcslen(greeting));
```

```
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
        break;
    }

    return 0;
}


//--------------------------------------------------------------------------
void    LoadDLL(void)
{
    // kernel32.dll containts GetSystemTimes() under XP
    hDllMod = LoadLibrary("dshow_webcam.dll");
    if( ! hDllMod )
    {
        //printf("LoadLibrary() failed\n");
        return;
    }

   // printf("LoadLibrary() OK\n");

    // note leading underscore
/*
    DllProc = (DLLPROC)GetProcAddress(hDllMod, "_FindCaptureDevice");
    if( ! DllProc )
    {
       // printf("GetProcAddress() failed\n");
        return;
    }
*/
}
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
void    UnloadDLL(void)
{
    if( hDllMod )
    {
        FreeLibrary(hDllMod);
    }
}
//--------------------------------------------------------------------------
```

C.4 Script - matdll.m

MATLAB script for testing the DLL:

```
% Matlab script for dshow_webcam.dll access.
% Created by Adam Cox
% Based on code provided by John Leis - USQ.
%
% Use the matlab commands for access:
% loadlibrary() - load dll
% calllib() - call dll function
% unloadlibrary(dllname) - unload dll
%
%
% List of dshow_webcam.dll accessible functions:
%
% FindCaptureDevice - Enumerate all cameras on the system
% WebCam1 - Camera 1 access
% WebCam2 - Camera 2 access
% GrabWebCamFrame1 - Grab frame from camera 1
% GrabWebCamFrame2 - Grab frame from camera 2
% ReleaseFilters - Release DirectShow and COM components

%-------------------------------------------------------------------------
% Example script
%-------------------------------------------------------------------------
clc
clear all
close all
%-------------------------------------------------------------------------

dllname = 'dshow_webcam'; % no .dll extension

if( ~libisloaded(dllname) )
    loadlibrary(dllname);
end

find = calllib(dllname,'FindCaptureDevice');

cam1 = calllib(dllname,'WebCam1');
cam2 = calllib(dllname,'WebCam2');

pause(1) % Allow pin to be accessed.

grab1 = calllib(dllname,'GrabWebCamFrame1');
%grab2 = calllib(dllname,'GrabWebCamFrame2');

load_imagel = imread('left0.bmp');
%load_imager = imread('right0.bmp');

imshow(load_imagel);

%rel = calllib(dllname,'ReleaseFilters');

%-------------------------------------------------------------------------
%if( libisloaded(dllname) )
%    unloadlibrary(dllname);
%end
%-------------------------------------------------------------------------

%close all
%clear all
%clc
```

C.5 Script - runlrc905640r150.m

MATLAB script for testing stereo processing:

```
%========================================================================
% Dense Stereo Processing - Living Room Test (Rectified) Logitech C905
% Resolution 640*480
% Baseline 150 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%========================================================================

clc;
clear all;

% Load images
iLt = imread('640_c905_left_rect_lr_150.bmp');
iRt = imread('640_c905_right_rect_lr_150.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:50];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 10);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.6 Script - runlrc905640r.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Living Room Test (Rectified) Logitech C905
% Resolution 640*480
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('640_c905_left_rect_lr.bmp');
iRt = imread('640_c905_right_rect_lr.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:10];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 10);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.7 Script - runlrc905640nr150.m

MATLAB script for testing stereo processing:

```
%========================================================================
% Dense Stereo Processing - Living Room Test (Not Rectified) Logitech C905
% Resolution 640*480
% Baseline 150 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%========================================================================

clc;
clear all;

% Load images
iLt = imread('640_c905_left_norect_lr_150.bmp');
iRt = imread('640_c905_right_norect_lr_150.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:50];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 10);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.8 Script - runlrc905640nr.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Living Room Test (Not Rectified) Logitech C905
% Resolution 640*480
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('640_c905_left_norect_lr.bmp');
iRt = imread('640_c905_right_norect_lr.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:10];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 10);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.9 Script - runbsc905640r150.m

MATLAB script for testing stereo processing:

```
%========================================================================
% Dense Stereo Processing - Book Shelf Test (Rectified) Logitech C905
% Resolution 640*480
% Baseline 150 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%========================================================================

clc;
clear all;

% Load images
iLt = imread('640_c905_left_rect_bs_150.bmp');
iRt = imread('640_c905_right_rect_bs_150.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:200];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.10 Script - runbsc905640r.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf Test (Rectified) Logitech C905
% Resolution 640*480
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('640_C905_left_rect_bs.bmp');
iRt = imread('640_C905_right_rect_bs.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:50];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.11 Script - runbsc905640nr150.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf Test (Not Rectified) Logitech C905
% Resolution 640*480
% Baseline 150 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('640_c905_left_norect_bs_150.bmp');
iRt = imread('640_c905_right_norect_bs_150.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:200];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.12 Script - runbsc905640nr.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf (Not Rectified) Test Logitech C905
% Resolution 640*480
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('640_C905_left_norect_bs.bmp');
iRt = imread('640_C905_right_norect_bs.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Stereo processing
shiftrange = [0:50];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.13 Script - runbsc200640r.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf Test (Rectified) Logitech C200
% Resolution 640*480
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iRt = imread('640_c200_right_rect_bs.bmp');
iLt = imread('640_c200_left_rect_bs.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:100];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.14 Script - runbsc200640nr.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf Test (Not Rectified) Logitech C200
% Resolution 640*480
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iRt = imread('640_c200_right_norect_bs.bmp');
iLt = imread('640_c200_left_norect_bs.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:100];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.15 Script - runbsc200320r.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf Test (Rectified) Logitech C200
% Resolution 320*240
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('320_C200_left_rect_bs.bmp');
iRt = imread('320_C200_right_rect_bs.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:100];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.16 Script - runbsc200320nr.m

MATLAB script for testing stereo processing:

```
%=========================================================================
% Dense Stereo Processing - Book Shelf Test (Not Rectified) Logitech C200
% Resolution 320*240
% Baseline 50 mm
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%=========================================================================

clc;
clear all;

% Load images
iLt = imread('320_C200_left_norect_bs.bmp');
iRt = imread('320_C200_right_norect_bs.bmp');

% Crop distorted edges
iLt=imcrop(iLt, [40,40, 600,440]);
iRt=imcrop(iRt, [40,40, 600,440]);

tic
% Apply stereo processing
shiftrange = [0:100];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange, 2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```

C.17 Script - runbenchmark.m

MATLAB script for testing stereo processing:

```
%==========================================================================
% Dense Stereo Processing - Book Shelf Test (Rectified) Logitech C200
% Resolution 640*480
% Script by Adam Cox - 2011
%
% Processing completed by:
%
% Stereo using diffuse connectivity (Matlab 7 code)
% Abhijit Ogale (ogale@cs.umd.edu)
% Computer Vision Lab, University of Maryland at College Park
% http://www.cfar.umd.edu/users/ogale
%==========================================================================

clc;
clear all;

% Load images
iRt = imread('l.bmp');
iLt = imread('r.bmp');

tic
% Apply stereo processing
shiftrange = [0:10];
[bestshiftsL, occlL, bestshiftsR, occlR] = stereoCorrespond(iLt, iRt, shiftrange,2);
toc

% Output Disparity Maps for left and right images
figure; imagesc(bestshiftsR)
figure; imagesc(bestshiftsL)
```