University of California, Irvine

Multi-User Interaction with Tiled Rear Projection Display Walls

THESIS

Submitted in partial satisfaction of the requirements for Donald Bren School of Information and Computer Science Honors Program

By

Kitty Ho

Thesis Committee:

Professor Aditi Majumder, Chair

Duy Lai

ACKNOWLEDGEMENTS

First and foremost, I offer my sincerest gratitude to my advisor, Aditi Majumder, who supported me for two years in the University of California, Irvine Computer Graphics Group. She offered me a chance to take a computer graphics graduate course, and introduced me to the graduate students in the computer graphics group. She also allowed me to work on this exciting project in the Calit2 Computer Graphics Lab. Without her help and support, I would not have this opportunity and experience on working on research under Donald Bren School of ICS Honors Program.

I also would like to thank Professor Gopi Meenakshisundaram, who exposed me to many opportunities in different projects in the computer graphics group including cell segmentation and automobile computer vision system.

I would like to thank all the graduate students, especially Duy Lai, in the Computer Graphics Research group who helped me through my research career and gave me so many ideas and advice.

ABSTRACT OF THE THESIS

Multi-User Interaction with Tiled Rear Projection Display Walls

By

Kitty Ho

Bachelor of Science in Information and Computer Science

University of California, Irvine, 2011

Duy Lai

Ph.D. in Computer Science

University of California, Irvine, 2010

Professor Aditi Majumder, Chair

Previous work has developed the first distributed paradigm for multiple users to interact simultaneously with large tiled rear projection display walls. The paradigm allows scalability across different 2D applications, interaction modalities, displays, and users. Earlier work utilizes a nine projector display to demonstrate four challenging 2D applications on the scalability of the method: map visualizations, virtual graffiti, virtual bulletin boards, and emergency management system. Our research group proposes a new challenge to this large tiled rear projection display wall, in which users are allowed to interact with 3D models on a six projector display. We present the challenges faced switching from 2D applications to 3D model display, and our improvements on the communication between hand gesture detection and the application response.

1. Introduction

Large display walls have been widely used in different applications such as visualization, entertainment, training, and simulation. Multi-projector display walls allow lower resolution projectors to present quality images on a large-scale display wall. This has brought our group's attention to create a scalable distributed paradigm for multi-user interaction with tiled rear projection display walls. In previous work, our research group has developed a distributed network of plug-and play projectors (PPPs). Each PPP consists of a projector, a camera, and a communication and computation unit, that is simulated by a computer[1]. It is used for the detection of hand gestures and communication among the PPPs in order to create a seamless multi-projector display that responds to a user's hand gestures.

Presenting 3D Models on a single 2D display has been a significant focus in different applications for the past decade. Our interactive multi-user display wall is not an exception. In this paper, we propose a new application that displays 3D models on the display wall which also allows users to interact with it. This thesis is broken down in to five sections: Previous Research, Main Contributions, C++ Network System Application, Results, and Conclusion to have further explanations.

Figure 1: This figure shows some of the applications in action that was developed in the previous research. There are four different 2D applications that are shown in this image which will be discuss later in this thesis.

2. Previous Research

Earlier work has focused on the following contributions:

1. Designing single program multiple data (SPMD), an algorithm that can easily scale to multiple projectors. In other words, adding and removing PPPs to reconfigure the display would not cause any changes to the interaction algorithm[1].
2. Developing four different 2D applications that work independently. Each application has different reaction to different hand gestures. This allows future work to develop different 2D applications, and also implies that we can create different 3D applications independently.
3. Designing a distributive algorithm to allocate the loads of handling multiple user interactions on to different PPPs.

2.1. Plug-and-Play Projector (PPPs)

The plug-and-play projector is consists of several components: AnEpson projector with a resolution of 1024 x 768, a camera covered by infrared an filter that acts as an "eye" for detecting hand gestures on the display wall figure 2, an infrared illuminator to detect the actual hand gestures from the images taken by the camera, and a wireless communication unit that allows each PPP unit to communicate with the others and share interaction, reaction information[2].

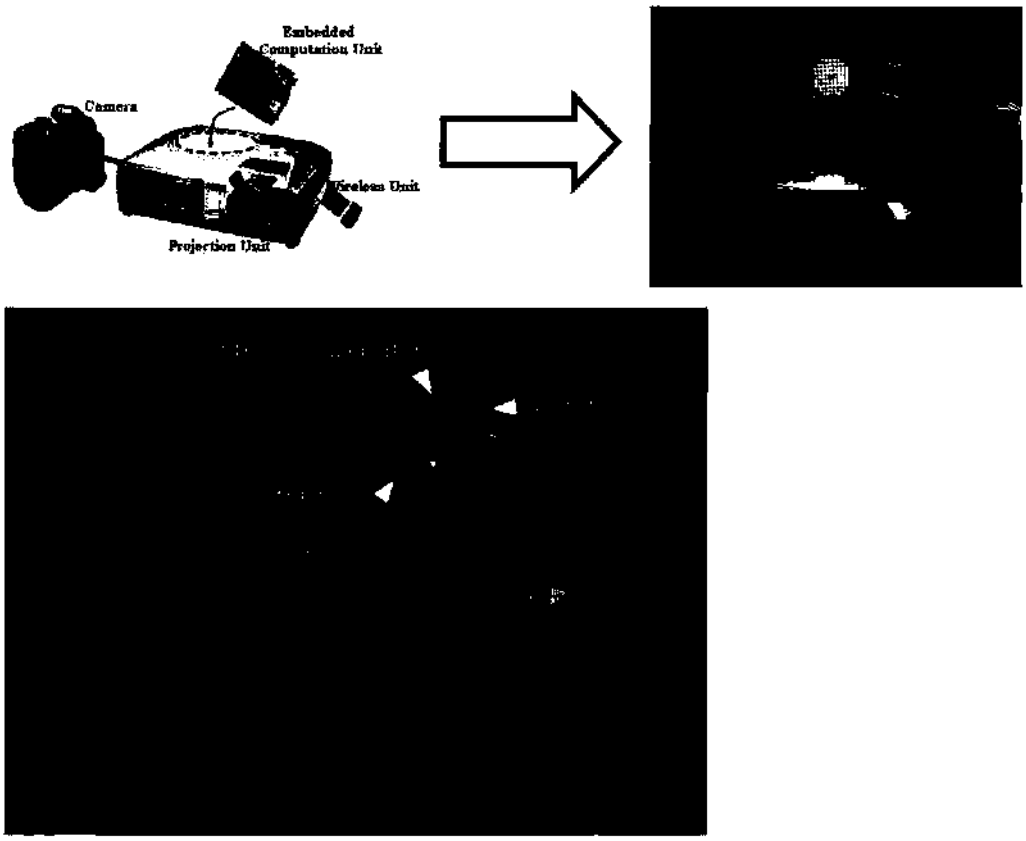


Figure 2: Top: The prototype PPP with a projector that contains communication and computation unit in it, a camera covered by infrared illuminator and an infrared illuminator. Bottom: The set up of a network of nine PPPs.

2.2. Single Program Multiple Data (SPMD)

A single program multiple data (SPMD) is a single program that runs on different nodes and performs similar tasks. Each PPP is a node of a homography graph [3], in which an edge is established between the nodes when the PPPs are adjacent to each other. If an edge exists between the two nodes, it means that they are going to communicate with each other for the interaction information. Each PPP contains a SPMD algorithm program that detects adjacent PPPs, and performs different actions depending on the hand gestures detected. Adding or removing PPPs will not affect the

interaction because the detection of the PPPs at the beginning would inform the PPPs about their adjacent PPPs. Therefore, the PPPs know the destination of the message they will send to when handling reaction information.

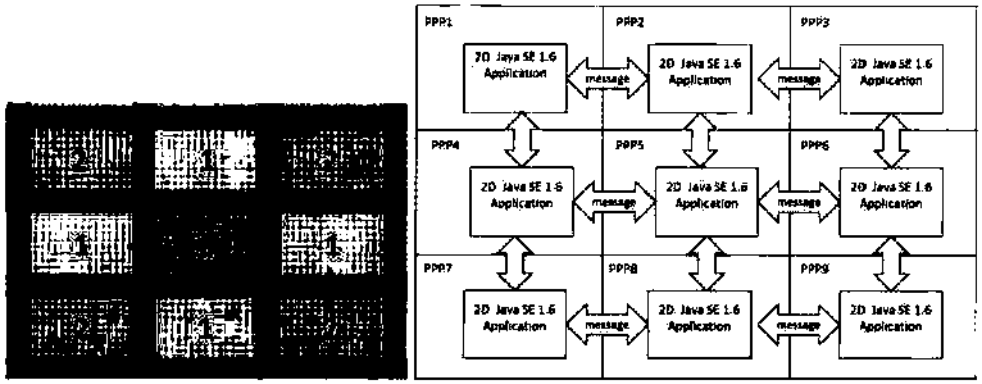


Figure 3: Left: The order of PPP node present in a homography graph. Right: Each PPP runs the same 2D Java SE 1.6. application and send messages to its neighbors.

2.3. Different Hand Postures

Our main goal for the display wall is to have multi-user interaction. Each application can define its own interpretation for each posture. Our applications use four main gestures: two fingers, one finger, open hand, and closed hand. In section 2.4, we are going to discuss what each hand gesture does in each application.

Figure 4: From left to right: Two Finger Posture, One Finger Posture, Open Hand, and Closed Hand.

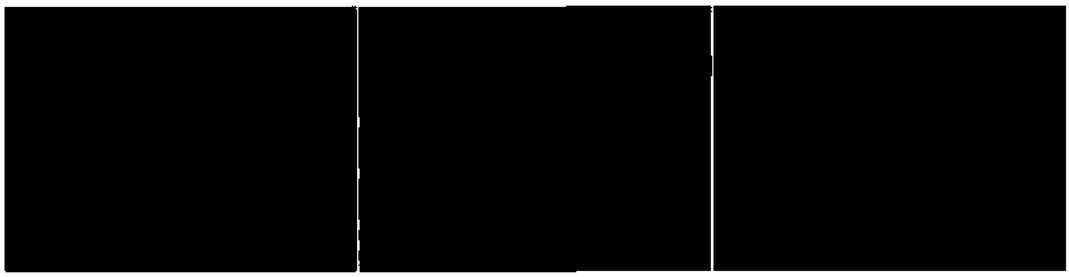


Figure 5: Images that the camera sees under infrared cover. From left to right: infrared background with no hand gesture detection, open hand, closed hand.

2.4. Four 2D Applications

This section introduces the four different applications that were developed and how the different hand gestures create different reactions.

2.4.1. Map Visualization

This application gets actual map data from the Google Map server, which allows users to view 2D Google Maps in real time on the display walls. Touching the screen with two fingers is used to change the displayed layer, and touching it with one finger is used to open and close, as well as change the size of individual working windows. Twisting an open hand on the display wall will change the zoom level of the map. Finally, any hand postures movements (up, down, left, right) can move the map.

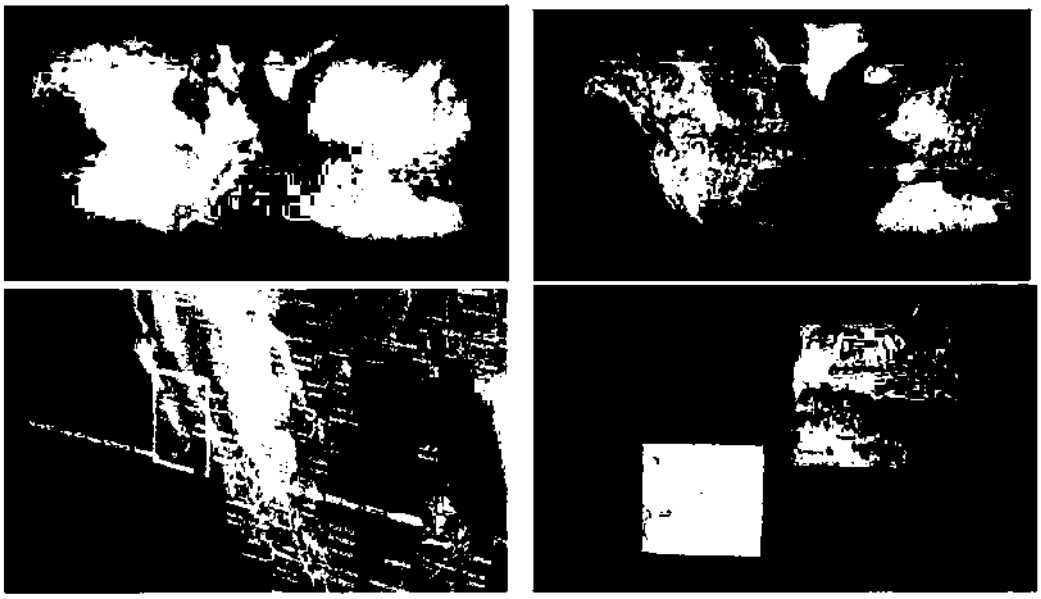


Figure 6: Top: Different views of Google Map on the display walls, changed by using two fingers. Bottom Left: Drawing a window on the map using one finger. Bottom Right: A zoom in view with two windows are drawn with different views of Google Map

2.4.2. Visual Graffiti

This application allows users to draw on the display walls using their fingers. Touching the screen with a finger, a closed hand, or an open hand is used to draw lines on the screen. The thickness of the lines depends on the thickness of the user's fingers and the size of the user's hand. Touching the screen with two fingers is used to bring up a color palette, wherein one finger can then be used to select a color from the palette.

Figure 7: Left: Drawing on the graffiti application using one finger. Right: Bring up a color palette with two fingers.

2.4.3. Visual Bulletin Board

This application is used for posting up different bulletins on the display walls. In our prototype, there are only a set number of notification types that can be posted on the board. Touching the screen with an open hand is used to load a new bulletin, or remove an existing bulletin. Touching an existing bulletin with one fingerwill either highlight or un-highlight the bulletin. Any of the hand gestures can be used to move the bulletin.

Figure 8: Left: Bulletin Board Application on the display wall. Middle: Highlighted bulletin using one finger. Right: Different size of bulletin after twisting a closed hand on the bulletin.

2.4.4. Emergency Management System

This application is used for allowing users to identify a route on the map during emergencies, and record the number of incident occurrences on the map. In our prototype, our map is set to the University of California, Irvine. Touching the screen with one finger allows user to create a route, open an options menu, and change an option in the menu. Touching it with an open hand is to delete part of the emergency route. Touching it with two fingers is used to create or delete an incident mark. Note that the map of the emergency management for this application does not move.

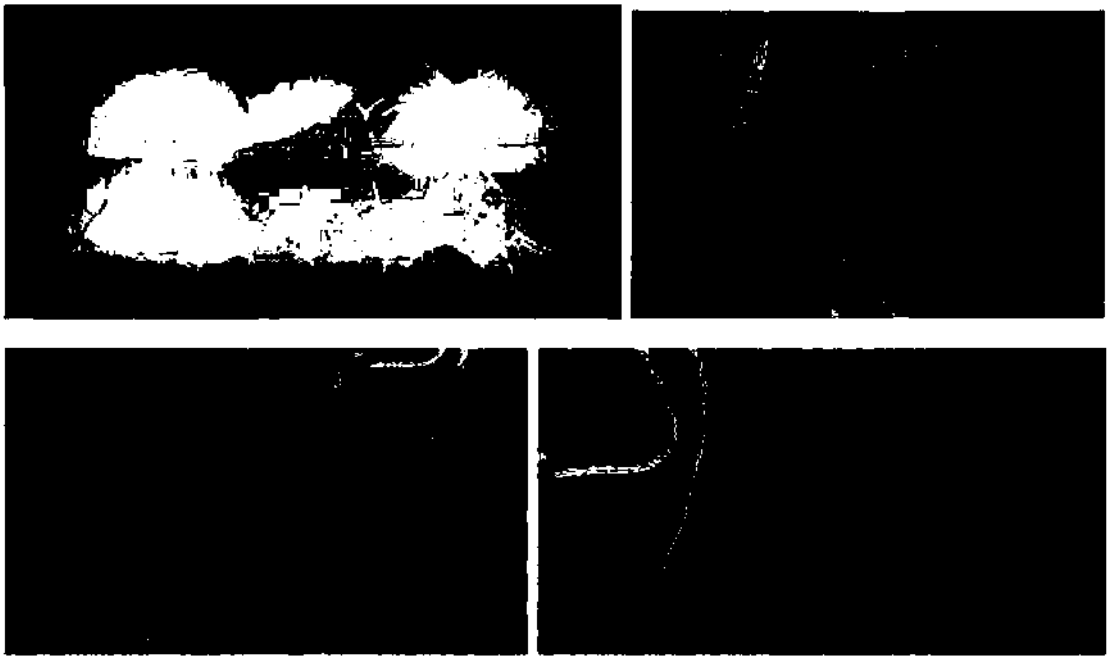


Figure 9: Top Left: Emergency Management System with University of California, Irvine map on it. Top Right: Creating an incident mark with two fingers. Bottom Left: Drawing an emergency route using one finger. Bottom Right: Changing the option menu using one finger.

2.5. Distributed Gesture Management

The distributed network of PPPs has no centralized server that manages the observed actions of the user. Therefore, each PPP is responsible for managing the actions that occur within its domain. In the previous work, when the gesture motion spans across multiple PPPs, the motion is tracked and information about the action is sent from one PPP to the next.

Action: When each PPP detects a hand gesture, it is defined as an action that contains a PPP identification number, gesture type (which includes the four postures mentioned in section 2.3), location on the display wall (x and y coordinate on the global coordinate system), and a timestamp. Each action is queued into a priority

queue according to the timestamp in which the PPP received the action of the hand gesture.

Event: Each PPP will go through the priority list queue and execute each action which creates an event for different application. An example of the event can be changing the zooming level of the map visualization, or creating an incident mark on the emergency management system. An event also considers the location of the action; if the location overlaps with another PPP, then the current PPP sends the action to the respective PPP in order for the other PPP to prepare for the action. Another use of location is to determine whether or not the hand gesture came from a single user or multiple users. If the hand gesture locations are close enough and the speed and acceleration of the previous gesture matches, then the event will continue with the previous event. For instance, in the graffiti application, it will continue drawing a line; otherwise it creates a new line at the new location.

2.6. Implementation

The detection of the hand gestures, laser-based interaction, camera image processing and recognition is performed in Matlab. The home-grown simple Matlab software detects the hand, computes its location, size and orientation, and determines its type by matching it to an existing hand posture library[1]. The distributed interaction framework is implemented using Java SE 1.6. Before the program starts, users should have already specified the number of PPP that will be used, and each PPP waits for a connection and establishes their relation with adjacent PPPs. Once the connection is

established, the PPPs start receiving messages from the Matlab software and perform reactions as explained in section 2.5.

3. Main Contributions

We use the PPPs and hand gesture detection in Matlab that was established in earlier work and build a new distributed interaction paradigm with 3D models on a 2D multi-projector display. For interaction, we propose a conversion on implementation from using Java SE 1.6 to C++ in Microsoft Visual Studio 2010. The highlights of our conversion are as follows.

1. In earlier work, we used Java SE 1.6 built-in library to set up the network connections. We introduce the POCO C++ Network System Library into our work in order to establish the communications between Matlab hand gesture detection and the C++ application.
2. Similarly, the communication among the PPP units needed to be reestablished using the same concepts.
3. Since we are working with 3D models, we need to incorporate the OpenGL library into our code. This allows us to apply multi-pass rendering in order to match the view frustum from each PPP onto one global coordinate system.

4. C++ Network System Application, conversion from Java SE 1.6

In this section, we describe the development of the C++ application and the usage of the POCO Library, mentioned in section 3, more in depth. Figure 10 shows the overview of the network system in the C++ application.

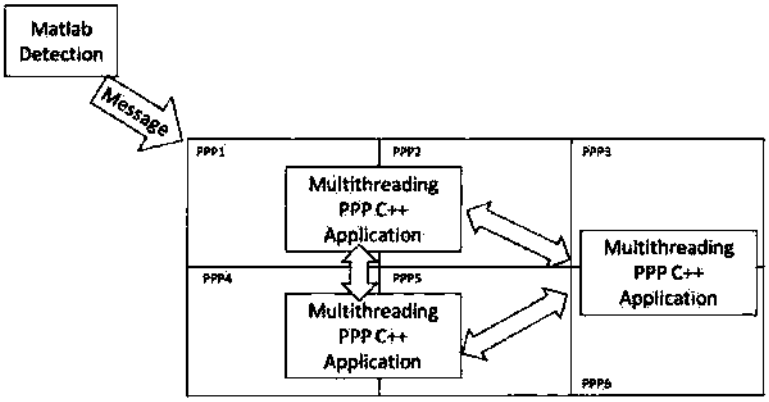


Figure 10: The overview of the C++ Network System Application

4.1. Network & PPP Information Configuration Files

A configuration file is commonly used to specify parameters that a computer program needs in order to correctly initialize itself.

We use the configuration files that were used in the previous Java SE 1.6 2D application. The configuration file contains the following information for each PPP:

1. PPP Identification number
2. IP Address for this specific PPP
3. Remote Port
4. Local Port
5. Alpha Mask
6. Quick Response Image
7. Quick Response Code Image
8. Monitor (PPP's resolution) width and height
9. Texture's width and height
10. Texture offset
11. Global Coordinate location of the screen
12. 3x3 Transformation Matrix

Each PPP has its own configuration file because each PPP has its own network connection IP address, remote port, local port, transformation matrix, and location. An example of the configuration is as follows.

```
0
touch1.calit2.uci.edu
37000
1231
wht1.png
qr1.jpg
qrc1.jpg

1024 768

1022 749

-694 -450

-29.000677 -28.853417
1024.000000 0.000000
1045.477908 772.140010
-0.000000 768.000000

1.060700 0.038676 723.020000
0.028188 1.069400 462.270000
0.000031 0.000009 1.024400

-692.877012 -440.958714
```

4.2. POCO C++ Network System Library

POCO stands for Portable Components, which is a collection of open source class libraries. This library allows us to build a network connection and to receive the message from Matlab via the User Datagram Protocol (UDP) to the C++ application.

4.2.1. Establishing a connection between Matlab and C++ application

1. First, we use the POCO library to create a socket address using the IP address that was given in the configuration file.
2. Next, we create a datagram socket that acts as a receiver that keeps listening for UDP packets sent from the Matlab software.

The code below achieves on building a connection and setting the C++ application up as a receiver, and having it wait for messages from the Matlab application.

```
Poco::Net::SocketAddress sa(Poco::Net::IPAddress(), port4);
Poco::Net::DatagramSocket dgs(sa);
char buffer[1024];
for (;;)
{
      Poco::Net::SocketAddress sender;
      int n = dgs.receiveFrom(buffer, sizeof(buffer)-1,sender);
      buffer[n] = '\0';
}
```

4.2.2. Multi-thread Application for PPP

In the previous work, each PPP needs to run its own application in order to establish connections and communicate with each other. Using the POCO Multithreading Library reduces the number of applications running simultaneously in half. There is a Runnable class in the POCO library which provides an entry point for a thread. We can add N number of PPPs on the display, but we only have two PPPs running on each computer. Therefore, in our application, we only create two threads for two PPPs.

The code below is a multi-threading application that sits on each computer connected to two projectors. The application has two threads, one thread per PPP. The program accesses it repeatedly that receives data coming from Matlab.

```
dataGramRunnable1 runnable1;
dataGramRunnable2 runnable2;
```

```
Poco::Thread thread1;
Poco::Thread thread2;

thread1.start(runnable1);
thread2.start(runnable2);

thread1.join();
thread2.join();
```

Figure 11: After the application has launched, two threads for each application become the listener and wait for Matlab and messages from other PPP's.

4.2.3. Communication among PPPs

When a PPP receives a message from Matlab, it needs to consider whether not an action needs to pass onto its neighbors. We establish the communication among the PPPs using an external POCO Network System Library as described in section 4.2.1 and a PPP neighbor configuration file; which is different from the configuration file mention in Section 4.1. The PPP neighbor configuration file contains the following information:

1. The total number of PPPs
2. Each PPP's IP address and port number that receives neighbor's message
3. The list of neighbors for each PPP

An example of the PPP neighbor configuration is as follows.

```
6
1 touch1.calit2.uci.edu 2222
2 touch1.calit2.uci.edu 2223
3 touch4.calit2.uci.edu 2222
4 touch2.calit2.uci.edu 2222
5 touch2.calit2.uci.edu 2223
6 touch4.calit2.uci.edu 2223
1 1 2 4 5
2 1 2 3 4 5 6
3 2 3 5 6
4 1 2 4 5
5 1 2 3 4 5 6
6 2 3 5 6
```

The program starts by parsing the two configuration files which allows the program to gather information from the PPPs, including the list of their neighbors. As mentioned in section 4.2.2, each PPP listens to the incoming messages from Matlab. The code below shows that it also listens to incoming packets from other PPPs, and then parses the packet and creates a timestamp for the packet when it arrives to organize the order of action. Finally, it sends the packet to its neighbors via specific port locations specified from the PPP neighbor configuration.

```
for(list<NeighborData*>::iterator it =
   secondPPP->getNeighborData()->begin();
   it != secondPPP->getNeighborData()->end();it++)
{
   NeighborData* nd = *it;
   Poco::Net::SocketAddress sa2(Poco::Net::IPAddress(), 0);
   Poco::Net::SocketAddress sa3(nd->ipAddress, nd->port);
   Poco::Net::DatagramSocket dgs2(sa2);
   dgs2.sendTo(buffer, sizeof(buffer)-1,sa3);
}
```

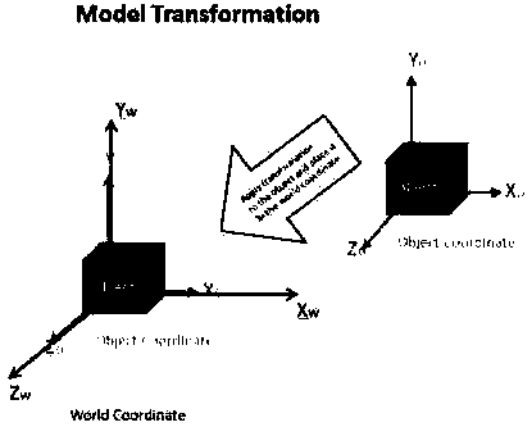
4.3.3D Model Rendering

This section goes over the basics of 3D models rendering and our approach on multi-pass rendering to achieve 3D models rendering onto the tiled rear projector display wall.

4.3.1. Basic 3D Models Rendering

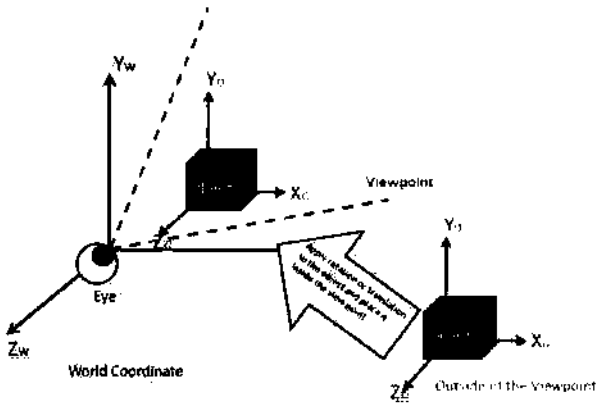
3D computer graphics composed of three steps: Modeling, Analysis, and Rendering. Modeling is the computer representation of the 3D world, analysis is to improve the quality and speed of the 3D object representations, and rendering is to generate 2D images of the 3D world; in other words, displaying 3D models onto a 2D plane. Our main purpose is to place 3D object or scene representations onto a 2D image representation. The following explains the four different steps which we called rendering pipeline to achieve our goal.

1. Model-view Transformation is to place the object to the world coordinate with the desired position, scale, and orientation.



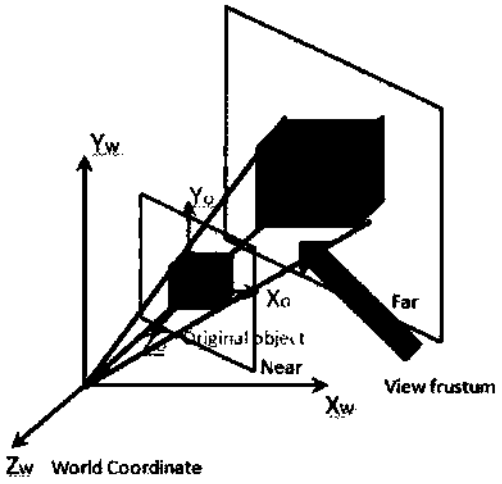
2. View Transformation is to place an object into the view point of the eye. The position of the eye should align with the origin with normal to the image plane with negative Z axis and view up vector with positive Y axis. The view transformation can be achieved by rotation and translation.

View Transformation



3. Projection transformation is to show the projection of an object on a farther plane. In other words, it is transforming the view frustum along with the objects inside it into a cuboid with unit square faces on the near and far plane and projecting the objects on the near plane.

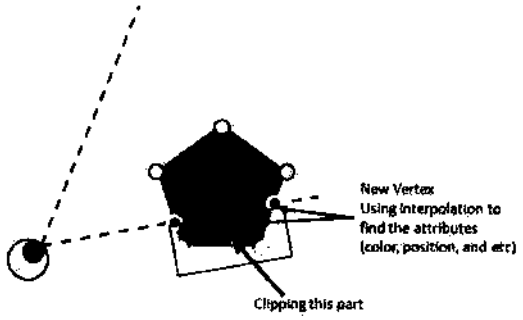
Projection Transformation



4. Clipping and Vertex Interpolation of Attributes

Clipping is to remove part of the object that is outside of the view frustum. When clipping an object, it creates new vertex on the boundary for the object and each vertex needs to have attributes. Interpolation is a way to find out the attributes of the new vertex by using the original vertex.

Clipping & Vertex Interpolation of Attributes



4.3.2. Multi-Pass Rendering

When rendering on 3D models on a 2D display, we called the region of space that we see as the view frustum. A large tiled projector based display is made up of multiple projectors. Such a display has a global view frustum that spans across all the projectors. In addition, each projector also has its own view frustum, which represents a small region of the global view frustum. Our goal for multi-pass rendering is to match each PPP's view frustum onto the global view frustum for the large display.

Given a transformation matrix which does a one-to-one pixel match from each PPP's local coordinate system to the global coordinate system. Each

PPP renders the small portion of the global view frustum. After the PPP renders its own portion of the image, it applies the transformation matrix onto the small image, which takes the image from local coordinate system to global coordinate system.

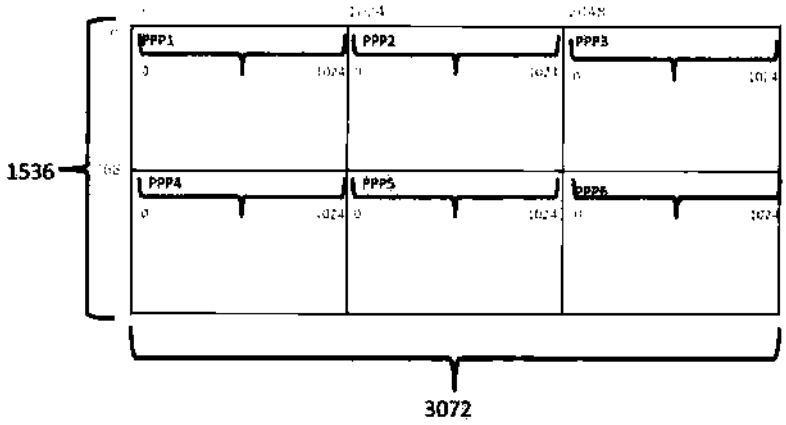


Figure 12: The overview of global coordinate system and local coordinate system. The blue is the global coordinate system of the large display. The red is the local coordinate system for each PPP. For instance, for PPP2, when local coordinate is 0, it is actually 1024 on the global coordinate system.

The multi-pass algorithm that provides the global coordinate points for each pixel:

Create an empty texture.

```
For each pixel on the local coordinate system
      Take the inverse of the given transformation matrix
      Multiply with the local coordinate pixel
      Normalize the result
      Copy the pixel data to the new texture at the
calculated position.
```

This algorithm is needed because the view frustum of the whole display is not a perfect rectangular shape to compensate the distortion of the projectors.

5. Results

We have prototyped a 3D city model collaborative application using this C++ conversion on our 8' x 4' display wall made of a 3 x 2 array of six projectors. The city model contains ten million triangles (about 4GB of data). Users can use a laser pointer to move from left to right or right to left to pan the city models respectively. Moving the laser pointer from down to up zooms-in the city model in the Z direction of the display wall and moving the laser pointer from up to down zooms out from the city models.

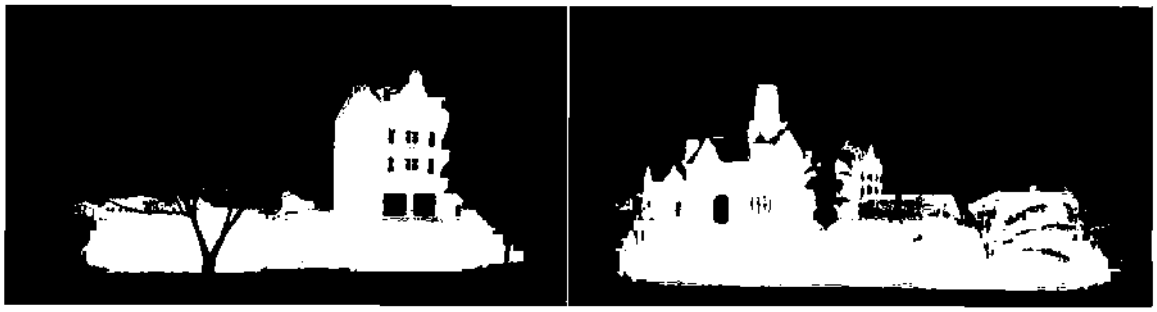


Figure 13: 3D City Models with ten million triangles are rendered on the 2D display wall. The right image is a zoomed out view of the image on the left.

6. Conclusion

Transitioning from a 2D system to a 3D system presents additional challenges that must be handles. Simply 2D gestures may take on a whole new meaning in 3D. New system architecture must be implemented to optimize system resources in order to maintain performance. System configurations and data layout must be redesigned to support more resource intensive applications.

To enhance performance, applications were written in C++; an external POCO C++ Library is utilized to to improve a couple things: (1) reduce the amount of applications running simultaneously for the PPP's in half, (2) converting Java SE 1.6 network

connection concepts into C++, and (3) allowing PPP's to communicate with each other using a different remote port. The multi-pass rendering allows 3D city models to match the view frustum of the 8' x 4' display and have each PPP display the right portion of the images including the overlapping part of the PPPs.

In the future we would like to extend our work to create more 3D applications. We believe that we can use hand gestures to move the city models on the display wall and instead of panning left and right, we would like to have rotation of the city models. Furthermore, we would like to allow users to create 3D models on the display wall.

Biblography

[1] Pablo Roman, Maxim Lazarov, Aditi Majumder, *A Saclable Distributed Paradigm for Multi-User Interaction with Tiled Rear Projection Display Walls,* IEEE Transactions on Visualization and Computer Graphics, 2010.

[2] Maxim Lazarov, Hamed Pirsiavash, Behzad Sajadi, Uddipan Mukherjee, Aditi Majumder. *Data Handling Displays,* IEEE/ACM International Workshop on Projector-Camera Systems (Procams)

[3] H. Chen, R. Sukthankar, G. Wallace, and K. Li. *Scalable alignment of large-format multi-projector displays using camera homography trees.* Proc. of IEEE Vis, pages 339–346, 2002.

[4] E. Bhasker, R. Juang, and A. Majumder. Registration techniques for using imperfect and partially calibrated devices in planar multi-projector displays. IEEE TVCG, pages 1368–1375, 2007.

[5] T. Grossman, R. Balakrishnan, G. Kurtenbach, G. W. Fitzmaurice, A. Khan, and W. Buxton. Interaction techniques for 3d modeling on large displays. ACM Symposium on Interactive 3D Graphics, pages 17– 23, 2001.

[6] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan. Distributed rendering for scalable displays. Proceedings of IEEE Supercomputing, pages 129–140, 2000.

[7] A. Khan, G. Fitzmaurice, D. Almeida, N. Burtnyk, and G. Kurtenbach. A remote control interface for large displays. ACM symposium on User interface software and technology, pages 127–136, 2004.

[8] S. Malik, A. Ranjan, and R. Balakrishnan. Interacting with large displays from a distance with vision-trackedmulti-finger gestural input. ACM symposium on User interface software and technology, pages 43–52, 2005.

[9] D. Stødle, Tor-Magne, S. Hagen, J. M. Bjørndalen, and O. J. Anshus. Gesture-based, touch-free multi-user gaming on wall-sized, highresolution tiled displays. 4th International Symposium on Pervasive Gaming Applications, PerGames, pages 75–83, 2007.

University of California, Irvine

Automobile Computer Vision System

THESIS

Submitted in partial satisfaction of the requirements for Donald Bren School of Information and Computer Science Honors Program

By

Kitty Ho

Thesis Committee:

Professor Gopi Meenakshisundaram, Chair

Jonathan Chu

ABSTRACT OF THE THESIS

Automobile Computer Vision

By

Kitty Ho

Bachelor of Science in Information and Computer Science

University of California, Irvine, 2011

Jonathan Chu

Bachelor of Science in Computer Science

University of California, Irvine, 2011

Professor Gopi Meenakshisundaram, Chair

At least one car accident occurs everyday, and is a major issue that we need to address. According to the car accident statistics released by the United States Department of Transportation, there are about 43,000 deaths because of car accidents every year. This is a significant number that greatly needs to be reduced. The most people, the majority of car accidents occur when the driver is not focused, but there also is a group of drivers who have trouble differentiating the color of traffic lights even if they focus. Drivers with protanipia, deuteranopia, protanomaly, and deuteranomaly may have trouble differentiating the colors of green and red, a big problem because these two colors are universally used for traffic lights. Our main goal is to reduce these types of car accidents as possible. For our project, we plan to build an embedded system that will improve driver safety, serve as a defense mechanism against car accidents and also be extrapolated to improve autonomous car development.

1. Introduction

The solution to help people differentiate the colors of the traffic lights is to build an automobile vision system which is a portable, easy to use car/camera system that will be able to read the color of traffic lights in real time, and relay that information to the driver with an audio message played back in the car, similar to a GPS. This device will take video of an intersection in real time with a camera connected to an embedded device which we called eBox. The software will then process the video stream to determine the color of the traffic light. Finally, the device provides an audio message to inform drivers of the color of the traffic light. The main purpose of this device is to assist drivers who are color blind, have color deficiency, and even normal drivers, to pay attention to the color of traffic lights by providing audio messages.

This solution is best implemented as an embedded solution due to the fact that the device needs to be portable so that the drivers can put it in their cars while driving. A software application for a mobile device can be developed for smart phones with the same idea but this will not be the best solution because not all drivers have smart phones. In addition, the quality of a smart phone's camera might not be good enough for video streaming in real time. Furthermore, video streaming and segmentation processes in real time uses a lot of memory which can slow down different applications that are running on the phone at the same time. Therefore, a separate embedded system that focuses on this ability is a better solution to the problem. We develop image processing software on a personal computer in Windows 7 Professional Platform to detect the color of circles through a web

camera in real time, and modify the Windows Embedded Compact 7 on a device called eBox. We will go into more details in the following sections.

2. Hardware Architecture

This section describes the hardware of the system, which includes the camera, a mini-computer (eBox), speakers, and display screen.

2.1. eBox-3310A-MSJK

The eBox is a compact computing device designed for applications where physical space is limited. It includes a VESA mount, 512MB DDR2 RAM to provide sufficient system memory support, Micro-SD slot, Compact Flash Slot, three high bandwidth USB 2.0 slots, two RS-232 ports, Mini PCI Socket, Audio, Wireless LAN, XGI Z9s video, and an Ultra-Low Power 1.0 GHz Vortex86DX System-On-Chip with integrated I/O peripherals. It supports Window Embedded CE 5.0, 6.0, and Compact 7.

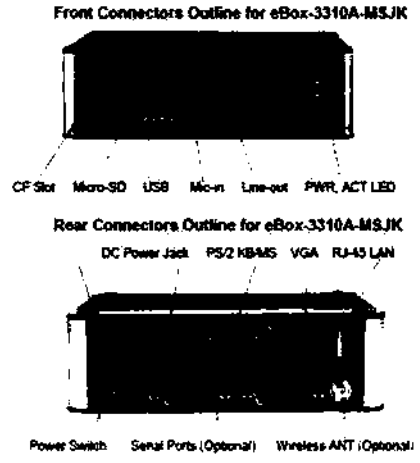


Figure 1: eBox Architecture

We believe that eBox is the best fit for our project due to the size (115 x 115 x 35mm) of the eBox is small enough to become a portable device in the car. It also supports Windows Embedded Operating System, which provides an easier way to develop image processing software in Windows Platform using C/C++.

2.2. Logitech Webcam Pro 9000

We use this specific webcam as our real time camera because it supports high-definition video up to 1600 x 1200, and take videos up to thirty frames per second. The system requires a Windows operating system and a dual-core CPU with one Gigabyte of random-access memory which fits the eBox specifications.

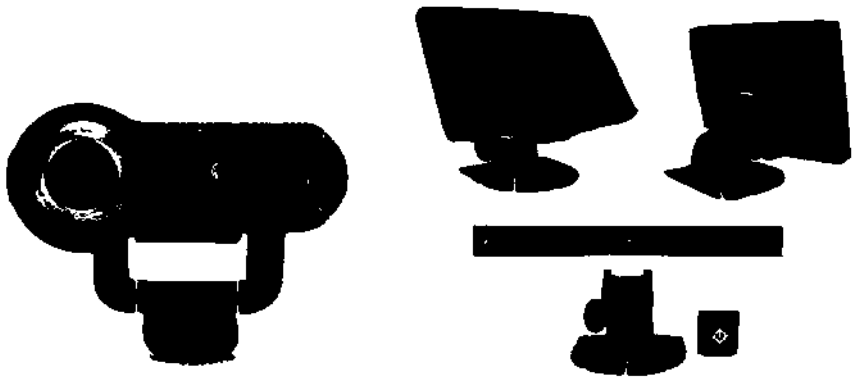


Figure 2: Left: Logitech Webcam Pro 9000 Right: Lilliput 7 VGA LCD 619.

2.3. Lilliput 7 VGA LCD – 619

This is a 7-inch TFT LCD touch screen with resolution of 800 x 480 that is used to display the output the Windows Embedded Compact 7 operating system and the image processing software. This LCD display also contains speakers for audio output

greater than or equal to one hundred megawatt to provide audio messages that are generated from the image processing software.

3. Windows Embedded Compact 7 Operating System

This operating system is a real-time operating system with ability to handle 32,000 concurrent processes and 2 Gigabyte memory footprint for each process. It provides variety of packages such as latest networking, multimedia, communication technologies, and etc. It also supports windows applications such as Internet Explorer, Windows Media CODECs, and so on. The highlights of generating a Windows Embedded Compact 7 Operating System Image are as follows.

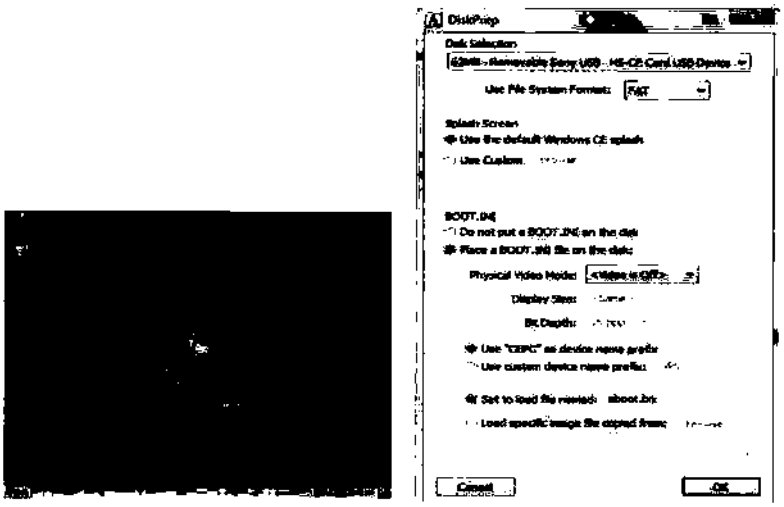


Figure 3: Left: Windows Embedded Compact 7 Right: DiskPrep Bootloader Generator

1. We use the required software such as Microsoft Visual Studio 2008, Windows Embedded Compact 7 CTP, My Window CE 7 SDK, AutoLaunch v100 x86 Compact 7, and CoreCon v100 Compact 7. Microsoft Visual Studio 2008 is the main component for the platform builder, the other software are there for plug-ins and

additional features for the OS Design to have more feature and fits the requirements of the eBox.

2. We configure the operating system by adding more components and drivers to it such as keyboard and mouse, FTP Server, ATAPI Storage driver, Z9s-XGI Display driver, VGA 1024x768 display driver, Ethernet driver, and USB ports driver.
3. We build and generate Compact 7 operating system runtime image by using Microsoft Visual Studio 2008 IDE with the Operating system design project being active and the result for the image file can be mounted onto the eBox device.

4. Image Processing Software

This section describes the functionality of software that is developed inside the processor. The program takes the video as an input, it segments the traffic light using a segmentation algorithm which will be discuss later in this thesis, and then perform color classification to determine whether a traffic light is green, yellow, or red.

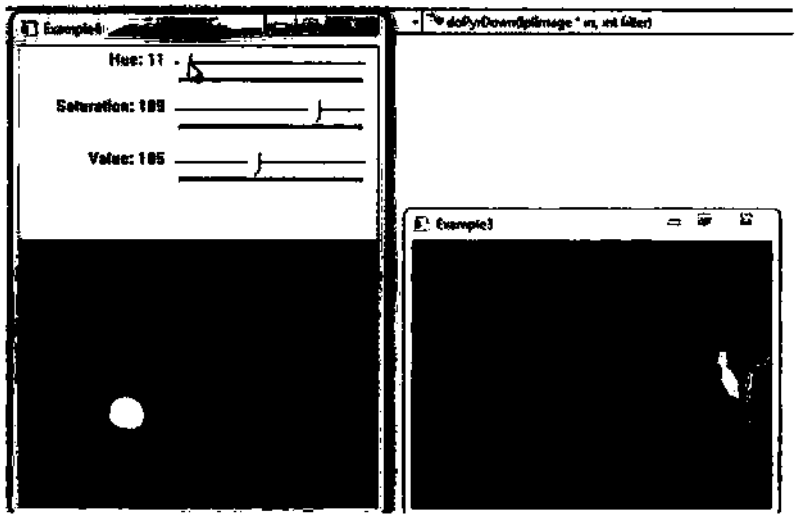


Figure 4: Our Image Processing Software Prototype with the color thresholding on the left and the video taken from the webcam in real time on the right. The red circle shows that the red circle from the webcam has been detected.

The three steps that we used for the software development are as follows:

1. Apply color thresholding using Open Source Computer Vision Library (OpenCV) to image to extract regions of interest. This allows us to provide a range of color for green, yellow, or red. In other words, we can block out any colors not of interest to allow the software to be more focused.

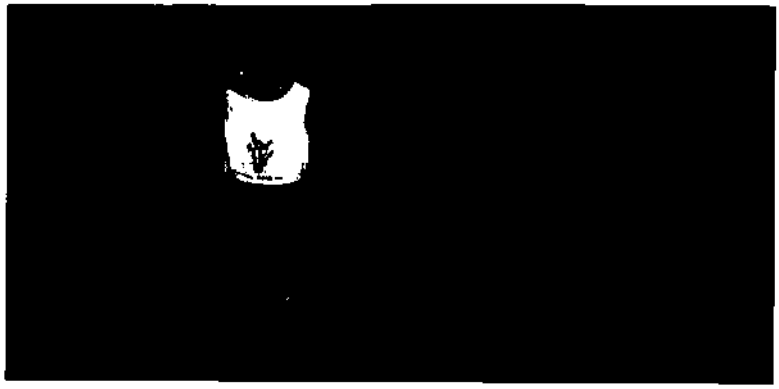


Figure 5: Color thresholding example which block out everything except red.

2. Apply Gaussian Blur to image to reduce noisy artifacts. Since we do not really need the details for every single pixel of the image, applying Gaussian blur would prepare the software to identify the shapes of the objects.

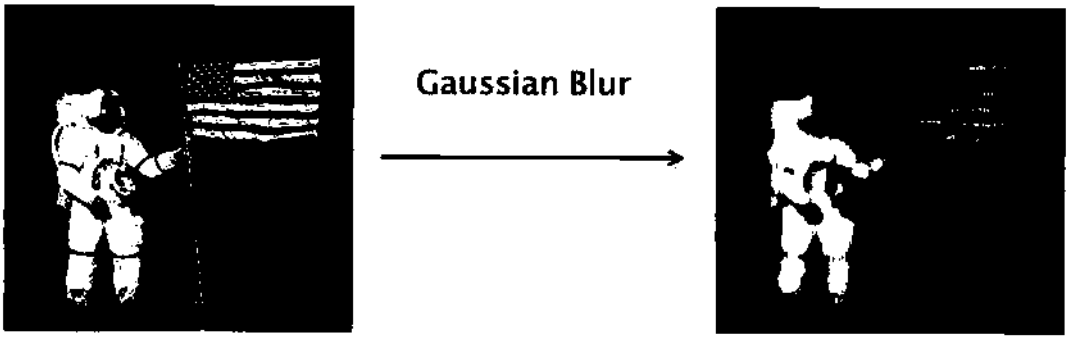


Figure 6: Gaussian Blur on a image

3. Apply Hough Circle Transform to detect circles. This algorithm will be discussed in more details in the next session.

4.1.Hough Circle Transform

In our software, we are only proposing to detector the colors of the circle stop lights and not the arrows, which usually direct drivers to turn left or right. Hough Circle Transform is used to determine the parameters of a circle when a number of points that fall on the perimeter are known.

We use points on the perimeter of a circle to draw a circle. After drawing circles on the perimeter, we will find that there is one point which all the circles from the perimeter would intersect and we know that is the center of the circle we are trying to detect.

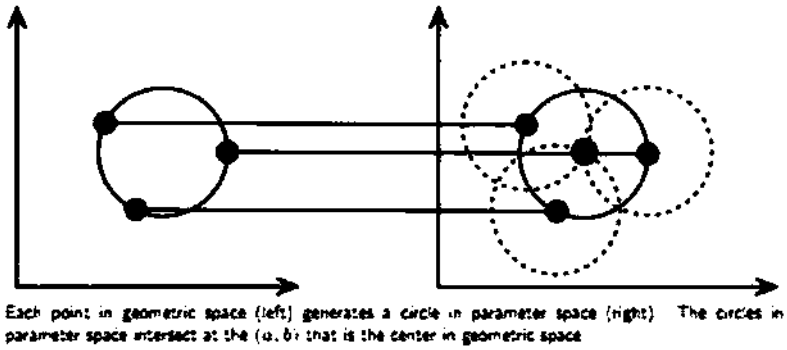


Figure 7: Hough Circles Transformation

5. Results

We finished implementing the image processing software to detect the color of the stop lights through a webcam in real time on a personal computer that runs in Windows 7

Professional using the methods described in section 5. We also finished design the Windows Embedded Compact 7 Operating System on the eBox.

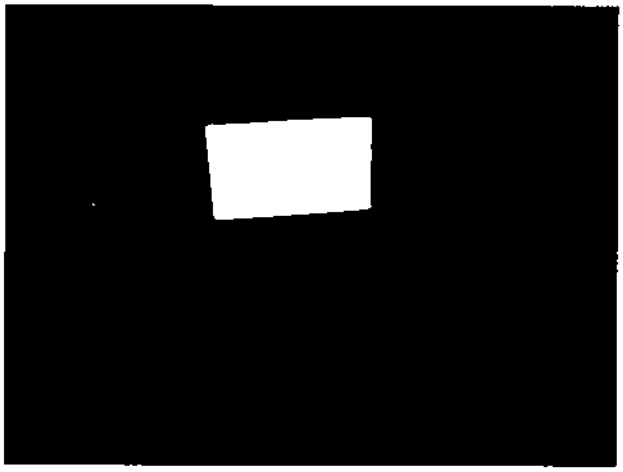


Figure 8: Our prototype Automobile Computer Vision System

6. Future Work

This project has not been completed yet, and the future work is as follows:

1. Writing a driver for the Logitech webcam for Windows Embedded System platform
2. Audio message needs to be recorded and embedded into the software after color recognition is complete.
3. Implement the recognition of arrows at turning signals.

7. Conclusion

Automobile Computer Vision System will be a great success in the future. It is the first in-car device that helps people with color blindness or color deficiency. By utilizing a simple architecture, the device can be bought by each family. This research can be invested as new product and increase the partnership with automobile industry and appreciation from the customers.

Bibliography

[1] Shapiro, Linda and Stockman, George. "Computer Vision", Prentice-Hall Inc. 2001

[2] Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Comm. ACM, Vol. 15*, pp. 11–15 (January, 1972)

[3] "Thresholding with an OpenCV Function", OpenCV Documentation, *http://myopencv.wordpress.com/2008/04/09/thresholding-with-an-opencv-function/*, April, 2008

[4] "Microsoft Windows Embedded Compact 7 Documentation", Microsoft, *http://www.microsoft.com/windowsembedded/en-us/develop/windows-embedded-compact-7-for-developers-overview.aspx*

[5] "eBox-3310A-MSJK", Intelligent Control on Processor, *http://www.embeddedpc.net/eBox3310AMSJK/*