WUCSE-2003-20

A Thesis on a 3D Input Device for Sketching Characters - Master's Thesis, May 2003

Authors: Schroering, Mark A.

April 17, 2003

SEVER INSTITUTE OF TECHNOLOGY

MASTER OF SCIENCE DEGREE

THESIS ACCEPTANCE

(To be the first page of each copy of the thesis)

DATE: January 17, 2003

STUDENT'S NAME: Mark A. Schroering

This student's thesis, entitled A Thesis on a 3D Input Device for Sketching Characters has been examined by the undersigned committee of five faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree Master of Science.

APPROVAL: ______________________________ Chairman

____

____

____

____

Short Title: 3D Input Sketching Device Schroering, M.Sc. 2003

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

A THESIS ON A 3D INPUT DEVICE FOR SKETCHING CHARACTERS

by

Mark A. Schroering

Prepared under the direction of Professor C. Grimm

A thesis presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Master of Science

May, 2003

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ABSTRACT

A THESIS ON A 3D INPUT DEVICE FOR SKETCHING CHARACTERS

by Mark A. Schroering

ADVISOR: Professor C. Grimm

May, 2003
Saint Louis, Missouri

The goal of this project is to develop a 3D input device using a stiff piece of paper and a camera. The camera tracks the piece of paper in 3D space. The user orients the paper in 3D space and then draws on the paper using a pen-like device. The camera tracks the movement of the pen on the piece of paper. The location of the pen in 3D space can then be calculated from the orientation of the paper.

A drawing application that uses this 3D input device was also developed. The application allows a user to make characters by sketching ellipses. The drawing application creates a virtual rendering of the paper and displays this to the user. As the user positions the real paper, the virtual one mirrors its movements. The user can draw shapes on the paper. These shapes then get rendered in the virtual scene.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis paper describes a general purpose 3D input drawing device. The 3D drawing device consists of a digital video camera that continuously tracks the position and orientation of a planar drawing surface. A rectangular piece of cardboard is used as the drawing surface. There are calibration markings on the cardboard that allow it to be tracked by the camera. Part of the drawing surface is transparent to allow a laser pointer to shine through. The camera also tracks the position of the laser pointer. The user can then draw on the paper using the laser pointer. The position and orientation of the paper determine where in 3D space the pen is drawing. This thesis also proposes a 3D shape sketching application that uses the device. The drawing application uses the input device to allow a user to sketch shapes in a virtual 3D scene. The user can see the virtual shapes that are being created on the computer monitor (see Figure 1.1).

The calibration pattern consists of a number of concentric ellipse pairs. The input device processes images from the camera to detect the border points of the ellipses. Equations for the ellipses are calculated using an ellipse fitting technique. These equations are then used as part of a minimization search to find the orientation of the digital camera relative to the paper. Once these parameters are found, the orientation of the drawing surface relative to the digital camera can be determined.

The sample drawing application uses the parameters from the input device to create a virtual scene of the drawing surface in 3D space. The user can move the paper in front of the digital camera and see the virtual paper move in the same manner. Drawing shapes is accomplished by using a laser pointer on the drawing surface. The 3D input device tracks the position of the pen. The drawing application uses this 3D data to create shapes in the virtual scene.

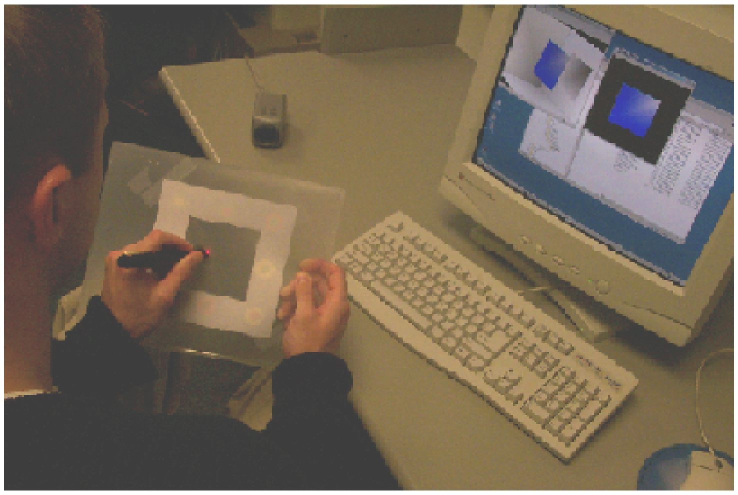

Figure 1.1: An illustration of a tracked sketching device made from a piece of cardboard, a calibration pattern, a digital video camera, and a laser pointer.

The goal of the project is to create a 3D input device that lends itself to drawing 3D characters. The device must be robust to different working conditions, including poor-quality cameras and changes in lighting. The calibration pattern must work when partially occluded or when part of the pattern is out of the camera's field of view. The device must also use inexpensive equipment that is available to most people.

The next chapter of this paper talks about previous work in camera calibration and spatially-aware user interface designs. Chapter Three describes the technique used to track the drawing surface. This chapter gives the details for the calibration pattern, the virtual camera model, and the calibration process. Chapter Four discusses the technique used for tracking the laser pointer. Chapter Five describes the 3D shape sketching application. Chapter Six lists the results and analysis from accuracy tests that were performed on the system. The last chapter discusses future work.

Chapter 2

Previous Work

2.1 Tracking a Plane

The problem requires the tracking of a drawing surface in the scene. This problem, and several variants, have been addressed in prior literature, most commonly in the context of camera calibration. A typical calibration pattern allows the camera to track the movement of the pattern's features. The most common calibration pattern used is a black and white checkerboard [17]. The corner points between the black and white squares are found, and the camera parameters are then calculated. This process is used in the Open Computer Vision (OpenCV) Library [11] to calibrate cameras[1].

The VISUAL PANEL [18] system tracks a flat cardboard pattern without using any explicit calibration pattern. Because a pattern is not used, a more complicated computational problem needs to be solved to determine the position and orientation of the piece of cardboard.

For this problem, we are primarily interested in capturing the position and orientation of a drawing surface with a known pattern. There are several techniques that have been proposed for this problem. One is a differential method which tracks continuous motions of a plane using spatio-temporal image derivatives [2]. Another uses simultaneous estimates of both the position of the plane and the texture pattern of that plane [3].

The technique that is most related to the solution described in this thesis is the single image direct methods of computing planar pose. Much of this work is

[1]OpenCV is a software library that is used for real-time computer vision applications.

summarized in [10], which integrates previous work into a unified geometric framework which uses combinations of point and line features.

2.2 Spatially-aware Devices

Non-traditional user interfaces, such as Graspable [5], Tangible [9], or Manipulable [7] interfaces, have been shown to be better than the traditional mouse and keyboard for many applications. In these alternative interfaces, the user picks up and manipulates a real 3D object. This takes advantage of the phenomenal human ability to grab, move, orient, and reason about 3D objects. Other examples of this type of interaction are interface props used for neurosurgical visualization [8] and BRICKS [6], a software and hardware framework for quickly prototyping graspable user interfaces.

Most position-aware devices have been implemented using specialized hardware, such as the Polhemus FASTRAK six degree-of-freedom trackers [14]. The device described in this thesis is not as general as the ones listed above, but it also does not require specialized hardware. This makes it more suitable for the home, school, or office environment.

Chapter 3

Drawing Surface Tracking

The core technical challenge of this project is the computer vision algorithm for tracking the drawing surface. These problems are theoretically well understood in the computer vision community. A calibration pattern is built with easily recognizable features. These features are then found in a 2D video image stream. The perspective transformation that takes 3D pattern features to 2D image pixels is then calculated.

Our requirement for this project was to come up with an algorithm that was robust to lighting conditions, poor-quality cameras, and partial occlusions. The pattern we use is built with color-contrasting ellipses. The 2D image features that need to be detected are the ellipses. Our application requires a minimum of three ellipses to be detected.

The input to this process is a video stream of the calibration pattern on the back of the drawing surface. Each image is scanned to find the pixels that lie on the ellipse borders. The equations of the ellipses are then calculated using these boundary pixel points. The ellipse equations are used in a minimization search to find the rotation and translation of the digital camera relative to the drawing surface. Once the rotation and translation values of the digital camera are known, the translation and rotation of the drawing surface can be determined.

Section 3.1 defines the calibration pattern. Section 3.2 describes the standard virtual camera model. Section 3.3 goes over the 2D feature detection algorithms. The search for the camera parameters is defined in section 3.4. Lastly, section 3.5 describes how the camera parameters are translated to paper movement.

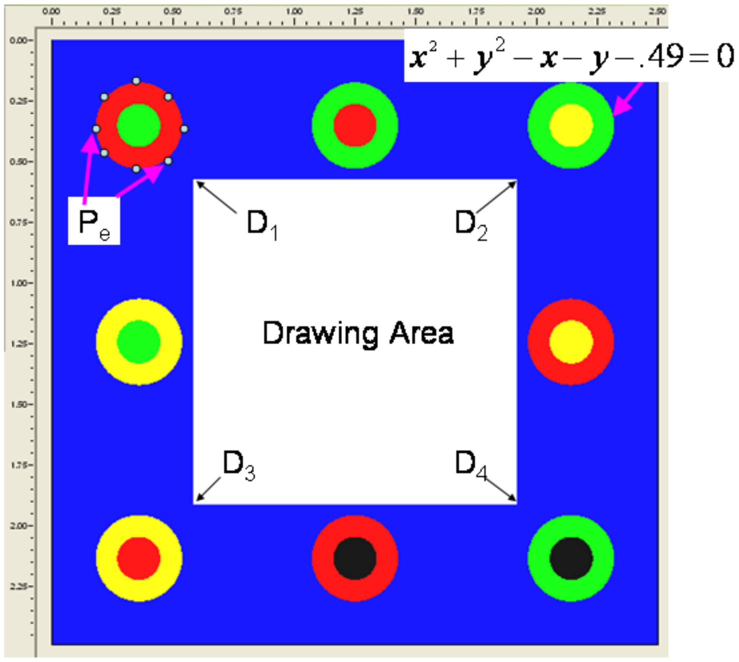

Figure 3.1: Calibration pattern with chromaglyphs. The points D_{1-4} are the corners of the drawing space. The points, P_e, are the known points on the ellipse borders. The circles in the pattern can be represented by equations of the form above.

3.1 Calibration Pattern

We considered several properties when designing the pattern. First, the pattern features must be easily distinguishable under different lighting conditions. Second, the pattern must be asymmetric, or different poses will have the same appearance. Third, the calibration pattern must work when partially occluded. The user holds the paper, so it is likely that the user's fingers will block some of the pattern. Also, the pattern may only be partially visible. Lastly, there must be sufficient space left on the drawing surface for sketching (see Figure 3.1).

To satisfy these constraints, the decision was made to use eight chromaglyphs [15] arranged around a drawing surface. The points D_{1-4} represent the corners of the drawing space. A chromaglyph is composed of N discs, each with a unique color, chosen from a set of M prototype colors [15]. For this application $N = 2$, and $M = 4$. The permutations of the disc color uniquely identify the glyph. The calibration pattern has eight chromaglyphs, positioned along the borders of the pattern. The pattern itself could be any size; for this application, we chose a size that could be easily handled by a user. The number of chromaglyphs was chosen so that only 1/4 of the pattern needs to be visible. The location of the chromaglyphs on the pattern allows a transparent drawing region in the center of the pattern.

3.1.1 Making the Calibration Pattern

We made a virtual version of the calibration pattern using the Open Graphics Library (OpenGL). The pattern can be saved to a bitmap file, which can be printed on a standard printer at any desired size. The pattern is cut out and glued to a stiff piece of cardboard.

3.2 Camera Model

The following is a list of definitions for terms that will be used throughout this paper.

- **Frame of Reference**: measurements are made with respect to a particular coordinate system called the reference frame.
- **World Frame**: a fixed coordinate system for representing objects in the world.
- **Camera Frame**: coordinate system that uses the camera center as its origin.
- **Image Frame:** coordinate system that measures pixel locations in the image plane.

The classic model for a camera is a pinhole at a fixed distance from an image plane. The basic assumption behind this model is that the relation between coordinates in the World Frame and coordinates in the Image Frame is a linear projection. This means that straight lines project to straight lines. This model does not take into account lens distortion. The resulting image is also inverted. This is not convenient for image analysis because the image needs to be inverted for proper processing. We use a model that is equivalent to the pinhole model, but the image is on the same side of the pinhole as the object (see Figure 3.2) .

There is a set of parameters that describe the characteristics of a camera. They are summarized in Table 3.1. The intrinsic parameters are those that specify properties of the camera itself. These parameters do not change unless the camera is re-focused and can therefore be calculated once using standard calibration techniques [17]. The extrinsic parameters describe the position and orientation of the

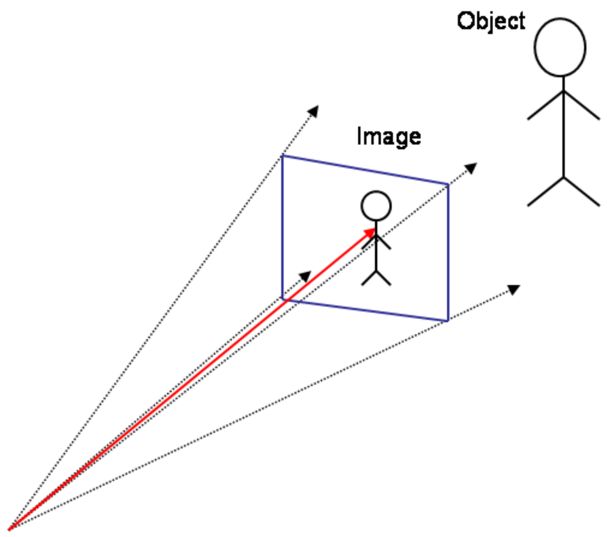

Figure 3.2: Camera Model

camera in the World Frame. These parameters change as the camera's position and orientation change.

Figure 3.3 provides an illustration of the extrinsic parameters, and Figure 3.4 shows the intrinsic.

We used an approximated guess for the intrinsic parameters instead of calculating the intrinsic parameters using OpenCV [11]. Calculating the intrinsic values would have required adding another calibration step to our process and a different calibration pattern that works with OpenCV. We were able to come up with an approximation that provided us with accurate results.

The camera model provides a function, $V(P)$, that converts points in the World Frame to points in the Image Frame. The function takes a point in the World Frame, P, as input and returns a 2D image point, q. The function uses a perspective transform, C, and a rotation plus translation matrix, $[R|T]$. The perspective, C, is intrinsic, and the $[R|T]$ is extrinsic. The following equations show how $V(P)$ transforms P to q:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = C[R|T]P \tag{3.1}$$

Table 3.1: Camera parameters

	Parameter	**Description**
Intrinsic	(u_0, v_o)	Center of the image projection, usually close to $(W/2, H/2)$, where W is the width of the image and H is the height.
	(α, β)	Represents the combination of the focal length, the aspect ratio, and the scale up to image coordinates.
	γ	The skew in the camera
Extrinsic	R	The camera's rotation matrix relative to the World Frame
	T	The translation of the camera relative to the World Frame represented as a vector.

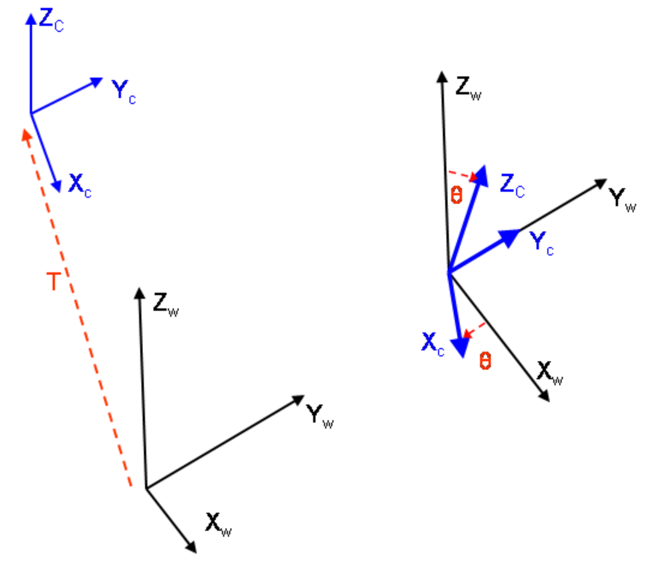

Figure 3.3: Extrinsic Properties of the Camera.

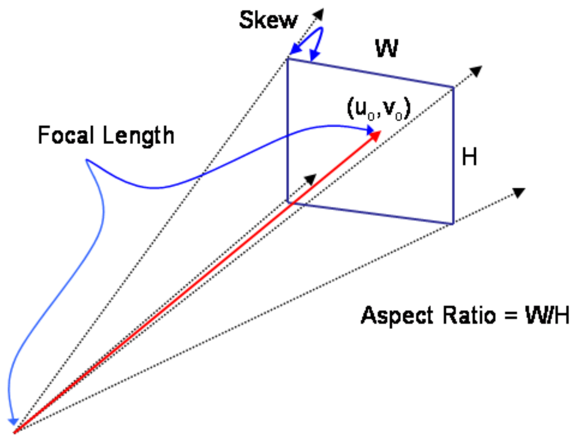

Figure 3.4: Intrinsic Properties of the Camera.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{x0} & R_{x1} & R_{x2} & T_x \\ R_{y0} & R_{y1} & R_{y2} & T_y \\ R_{z0} & R_{z1} & R_{z2} & T_z \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \tag{3.2}$$

$$q = (u/w, v/w) \tag{3.3}$$

The extrinsic parameters map a point in the World Frame to a point in the Camera Frame. The intrinsic parameters map a point in the Camera Frame to a point in the Image Frame. Figure 3.5 illustrates the relationship between the three reference frames.

3.3 2D Feature Detection

In this section, we describe how to find the features in the 2D image. The projection of the pattern causes the 3D circles to project as ellipses. The image is scanned to find the pixels that lie on the ellipse boundaries. The color of the pixels along the ellipse borders are used to determine which ellipse was found. The border points are then passed to an ellipse-fitting routine.

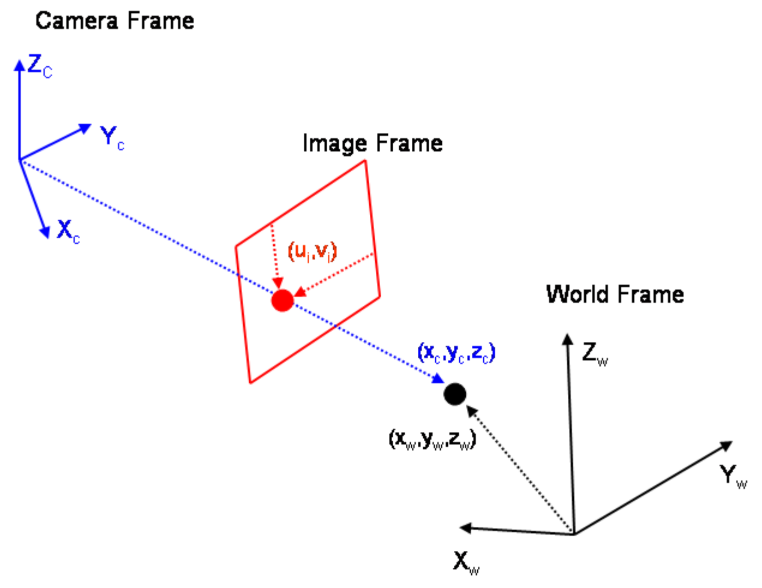

Figure 3.5: Relationships between reference frames.

3.3.1 Image Scanning

Image scanning is used to find the points in the image that fall on the borders of the chromaglyphs. The scanning classifies features in the image by color ratios. The colors chosen for the chromoglyphs are located in different corners of the color space [15]. Each color has a maximum or minimum concentration of red, green, or blue. Because of this, the colors are still distinguishable under different lighting conditions. For example, for a red pixel, the portion of red will always be greater than the portions of green or blue.

The scanning software acquires a buffer of pixel color values from the digital camera. This buffer is a snapshot of what the camera currently sees in its field of view. Each pixel in the buffer is then assigned one of the five prototype colors (red, green, yellow, black, or blue). If the pixel color cannot be determined, it is labelled as unknown.

The scanning software picks a pixel that is one of the four prototype colors as the starting point for a flood-fill search. The search returns a connected region of pixels that lie on the border of the ellipse. For example, when a red pixel is found, the flood-fill search will look for all of its neighboring red pixels that border blue pixels (see Figure 3.6).

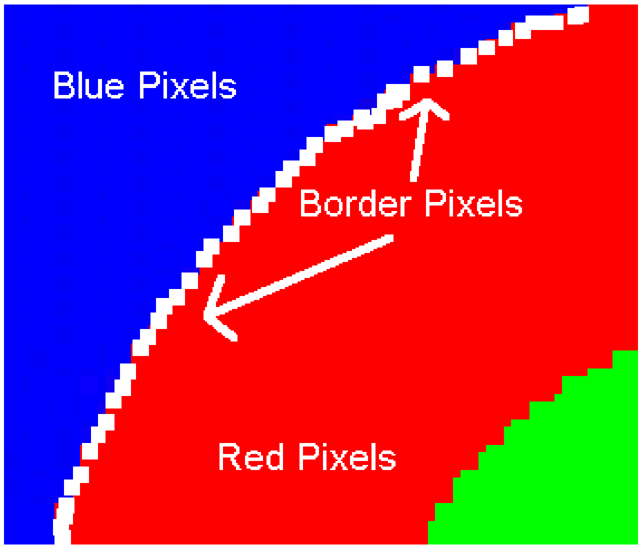

Figure 3.6: The red pixels that border the blue pixels are detected.

Since low-resolution cameras are being used, blurring can occur along the ellipse borders. This causes some of the pixel colors to be unidentifiable. The color value for these pixels is labelled ‘unknown’. When one of these unknown pixels is found during the flood-fill search, a color check of the unknown pixel’s neighbors is performed. If one of the unknown pixel’s neighbors is a different color, then it can be assumed that this pixel lies on the border. This technique allows for a thin line of blurring along the ellipse borders.

We first search for the outer border points. When the flood-fill search is complete, the center of the ellipse is calculated using the outer border points. This is also the center of the inner ellipse. The color of the inner ellipse is then identified by the center point’s color. The color of the outer and inner discs provide the identity of the chromaglyph. A flood-fill search is then performed on the inner circle to obtain the inner ellipse border points.

The entire image is scanned in this fashion to identify all of the chromaglyphs that are in the field of view. When the scanning is finished, the border points for the visible ellipses are passed to the ellipse fitting algorithm. Table 3.2 lists the pseudocode for the image scanning algorithm.

3.3.2 Ellipse Fitting

Once an ellipse’s border points have been identified, we can fit an ellipse to them. There are many approaches to ellipse fitting. The method used in this application is

Table 3.2: Image Scanning Algorithm

```
for each pixel in image do
  if pixel is one of the Prototype Colors AND pixel has not been scanned then
    Set scanned flag for this pixel to TRUE
    Outer border points = Perform flood-fill search on pixel setting scanned
    flag for each pixel scanned to TRUE
    Centroid = Calculate the centroid of the outer border points
    Inner border points = Perform flood-fill search on Centroid setting scanned
    flag for each pixel scanned to TRUE
    Glyph ID = permutation of pixel color and Centroid color
    Store inner and outer border points for the glyph identified by ID
  end if
end for
```

the one developed by Fitzgibbon, Pilu and Fisher [4]. The method finds the coefficients to the implicit equation of a conic:

$$F(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{3.4}$$

The equation above can also be expressed as $F(\overline{a}, \overline{x}) = \overline{a}\overline{x}$, where $\overline{a} = [a\ b\ c\ d\ e\ f]^T$, and $\overline{x} = [x^2\ xy\ y^2\ x\ y\ 1]^T$. The method finds the coefficients of the conic equation by solving the following eigenproblem:

$$D^T D\overline{a} = S\overline{a} = \lambda C\overline{a} \tag{3.5}$$

where $D = [\overline{x}_1\ \overline{x}_2\ ...\ \overline{x}_n]^T$. The vectors, $\overline{x}_{1...n}$, are the different observation points. In our case they are the eight points on the ellipse borders. S is the scatter matrix, $S = D^T D$, and C is the ellipse constraint, $b^2 - 4ac = -1$ in matrix form:

$$C = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.6}$$

The above eigenproblem must meet the following equality constraint:

Table 3.3: Ellipse Fitting Algorithm [4]

```
–Build Design Matrix
for each point on the ellipse border do
    D[i] = [pt.x * pt.x, pt.x*pt.y, pt.y*pt.y, pt.x, pt.y, 1]
end for
–Build scatter matrix
S = $D^T$ * D
–Build 6x6 constraint matrix
C = matrix in equation 3.6
–Solve generalized eigen-system
eV = SolveEigenSystem(S,C)
–Find the eigen values that satisfy $b^2 - 4ac < 0$
for i = 0 to 6 do
    if $eV[i][1] * eV[i][1] - 4 * eV[i][0] * eV[i][2] < 0$ then
        return eV[i]
    end if
end for
```

$$\overline{a}^T C \overline{a} = 1 \tag{3.7}$$

The eigenproblem has three non-trivial solutions, but only one satisfies $\overline{a}^T C \overline{a} = 1$. The eigenvalues and eigenvectors that solve the eigenproblem can be processed to obtain the solution for $\overline{a}$ [4].

The fitting is performed on each set of border points, which produces up to 16 sets of ellipse equation coefficients. Only the ellipses that are visible in the current image are used. Thus, there can only be up to 16 used at one time. These equations are used in the camera parameter search stage. Table 3.3 lists the pseudocode for the ellipse fitting algorithm.

3.4 Camera Parameter Search

The final step in the camera calibration process is the search for the camera model parameters that map the 3D circles ($z = 0$) to the ellipses found in the image. We know the points, P, on the 3D circles, and pick a subset of them to use for the search. We used eight points per ellipse (see Figure 3.1).

As the drawing surface is oriented in front of the camera, the calibration pattern that the camera sees gets warped. The 3D border point, P, can be mapped to a 2D image point, q with the following equation:

$$q = V(P) \tag{3.8}$$

where V is the camera model function described in section 3.2. If T and R represent the camera's true orientation, then the points should lie close to the ellipses found in the image scanning step. This means that the ellipse equation, $F_e(q_x, q_y)$, should evaluate to a value that is close to zero when using these points.

The goal of the search is to find the values of T and R that minimize $(F_e)^2$. We created an error function, $E(T, R)$, that we are trying to minimize. The function takes values of T and R as input. For each ellipse that was visible in the image, the function converts its corresponding border points in the World Frame, P_e, to 2D points in the Image Frame. The corresponding ellipse function $F_e(q_x, q_y)$ is then evaluated for each of these converted 2D points. The results are squared and summed for all eight points. The equation for $E(T, R)$ is as follows:

$$E(T, R) = \sum_{e=1}^{n} \sum_{i=1}^{8} [F_e(V(P_{ei}))]^2 \tag{3.9}$$

where $n \leq 16$ is the number of ellipses found in the image scan and P_{ei} is the i^{th} point on the e^{th} ellipse. We require a minimum of three and use a maximum of eight. More than eight could be used, but eight was chosen to reduce computation time. The search begins with a guess at the values of T and R. If $E(T,\ R)$ does not evaluate to a value that is close to zero, then T and R need to be modified in some way. A Simplex Search [13], also known as the Amoeba Search, is used to minimize $E(T, R)$. For the Simplex Search, T is represented as a translation vector $[x, y, z]^T$. R is represented as three rotation angles around the x, y, and z axes of the World Frame.

A Simplex Search is a multidimensional minimization, which means that it finds the minimum of a function of more than one independent variable. In our case we have six independent variables, the three components of the vector T, and the three rotation angles that make up R. The Simplex Search uses the *downhill simplex method* which requires only function evaluations, not derivatives. A Simplex can be thought of as a geometric figure in N dimensions, consisting of $N + 1$ points. In our case, $N = 6$, for the six input variables we have for $E(T, R)$. The algorithm starts by taking an initial guess of $N + 1$ points. These points define the starting shape of the simplex. In our application we use our initial guess for T and R as the starting point. We then create N more points by adding a constant to each independent variables. We now have a vector of size $N + 1$ that contains the initial points for the simplex. The algorithm then tries to find the minimum in the N-dimensional topography [13].

The downhill simplex method takes a series of steps to find the minimum of the function. Most of these steps involve moving the point of the simplex where the function is the largest through the opposite face of the simplex to a lower point. These types of steps are called reflections. When possible, the method expands the simplex in one direction or another to take larger steps. When it reaches a valley floor, the method will contract the simplex in the transverse direction and try to squeeze through the valley. This type of behavior is why the method is often referred to as the Amoeba Search [13].

The termination criteria for the search is when the vector distance moved in one of the steps is less than some tolerance value. In our application, we use a tolerance value of 0.000001. The number of iterations needed to find the minimum in the initial search is approximately one hundred. Subsequent searches are started using the previous values for T and R; they converge with about thirty iterations. If the display surface's orientation does not change much, the searches will take about ten iterations. Table 3.4 lists the pseudocode for the Simplex Search.

3.5 Calculating the Drawing Surface's Rotation and Translation

When the values for T and R are found, they represent the position and orientation of a movable camera in the World Frame. The drawing surface is assumed to be located at the origin of the World Frame. In the real world, the camera is stationary and the drawing surface is moving. We want to know the position and orientation of the drawing surface in the World Frame. We had to devise a way to use T and R to determine the position and orientation of the drawing surface. This problem is easier to explain through the use of an example.

Figure 3.7 illustrates a scenario that can occur in our application. In the real world, the drawing surface is rotated by θ degrees. When we apply our camera calibration technique, we obtain values for T and R that represent the scenario in the

Table 3.4: Simplex Search Algorithm [13]

```
while TRUE do
  for i = 1 to Number of points in simplex do
    Determine which point is the highest (worst), next-highest, and lowest
    (best) by evaluating the function for each point.
  end for
  Compute the fractional range from the highest to the lowest point and see if
  it is below the tolerance value
  if Range < Tolerance then
    return the lowest point in the simplex
  end if
  Extrapolate highest point in the simplex through the opposite face
  if Reflected point < Current Lowest Point then
    Try an extrapolation by a factor of 2
  else if Reflected Point > Current Lowest Point then
    Do a one-dimensional contraction from the high point
    if Contracted Point > Highest Point then
      Unable to get rid of high point, so contract around the lowest point
    end if
  end if
end while
```

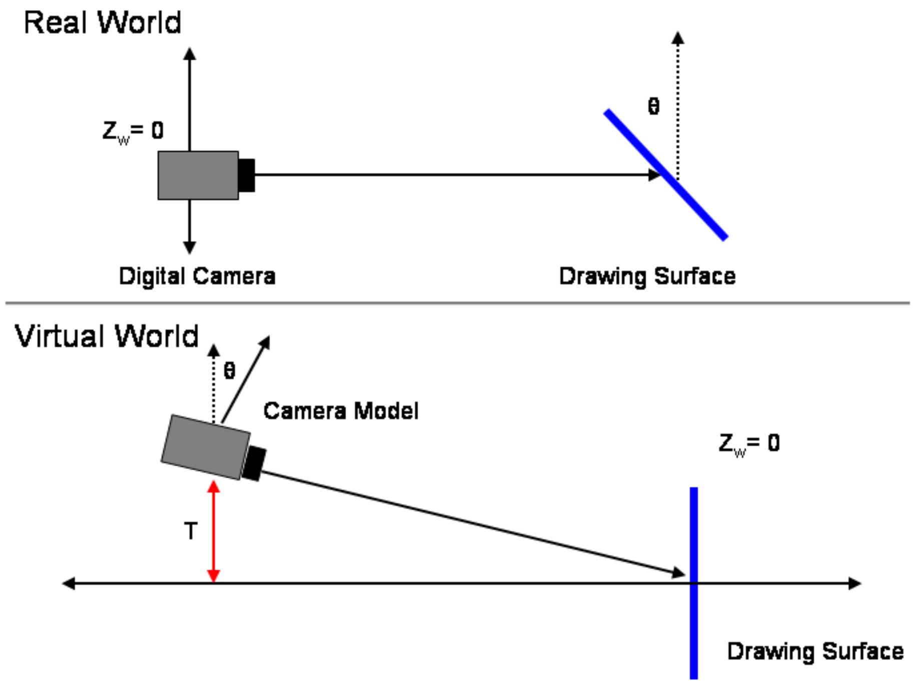

Figure 3.7: The relationship between a change in the real drawing surface's orientation and the virtual world.

lower half of the figure. The camera model is rotated by the same θ degrees that the drawing surface was in the real world. The rotation is just in the opposite direction. As one can see from the figure, the camera model is also translated to a new position in the World Frame.

To calculate the orientation of the drawing surface in the virtual world, we simply have to take the inverse of the matrix defined by R. However, we cannot apply $-T$ to the drawing surface. From the example above, we can see that this will apply a translation to the virtual drawing surface's position in the World Frame. The only movement that occurred in the real world was a change in the drawing surface's orientation. There was no change in the position.

To handle this situation, we calculate the change in the camera model's focal point. If the drawing surface is translated in the real world, the focal point of the camera model will change by the same amount in the virtual world. Before the Simplex Search is performed, we save the original focal point, f_{orig}. The Simplex Search is then executed and the new focal point, f_{new}, is retrieved from the camera model. A vector that represents the translational position of the drawing surface in

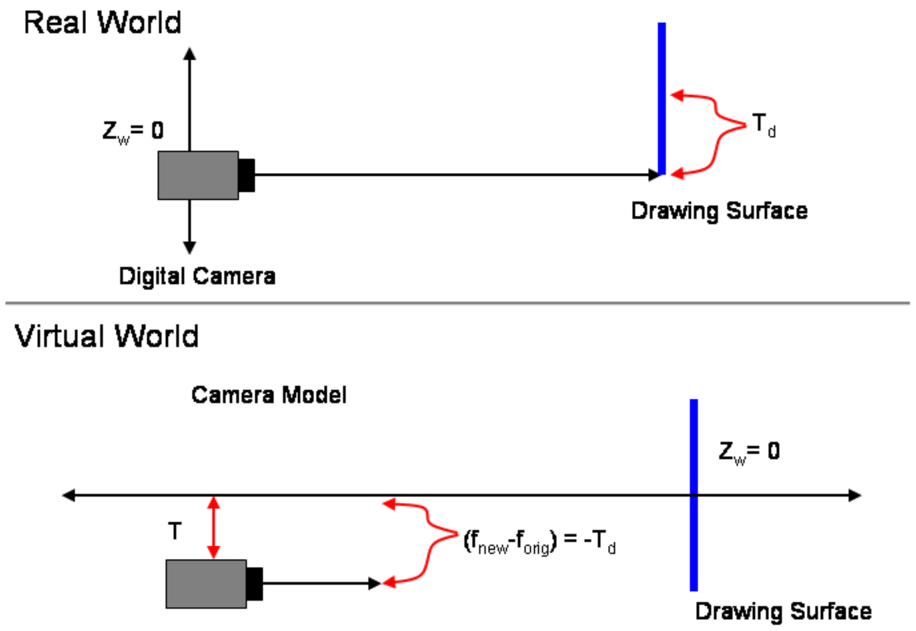

Figure 3.8: The relationship between a change in the real drawing surface's position and the virtual world.

the world frame is then equal to $-(f_{new} - f_{orig})$. We will refer to this translation vector as T_d. Figure 3.8 illustrates this scenario.

We now have a way to describe the position and orientation of the drawing surface in the World Frame. A rotation plus translation matrix that defines this transformation can be expressed as $[R^{-1}|T_d]$. Which can be expressed as $[R^T|T_d]$ since R is orthonormal.

3.6 Some Lessons Learned

Initially, we tried representing R as a quaternion in the minimization search. Quaternions are not the best representation for searching. They have the property of uniqueness, but there is an interdependence in the coordinates because of the normalization step. The choice was then made to use 3 rotation angles. The angles do not have the uniqueness property, but they are independent of each other. This property was needed in order to perform a successful search.

The translation error is also not uniform in all dimensions, as can be seen by examining equation 3.3. Note that the image point changes as a function of $x, y, 1/z$, since w is a function of z. One other thing to note is that the rotation and translation are linked under normal movements. Positioning the camera relative to the drawing surface results in a transformation that is both a rotation and a translation.

Chapter 4

Laser Pointer Tracking

We use a red laser pointer to sketch on the display surface. The inner region of the display surface is transparent. This allows a laser pointer to shine through and be detected by the camera. To indicate an input action, the user positions the drawing surface at the desired location, and then turns on the laser pointer so that it is shining through the transparent region. The image scanning will find the location of the laser pointer, as the centroid of the red spot, in 2D image coordinates. This location is then converted to a 3D world coordinate relative to the current position of the drawing surface. This chapter describes the procedure for how to track the pointer in 2D image space and how to calculate the 3D input location.

4.1 Finding the Laser Pointer in 2D Space

In addition to looking for ellipse boundary points, the image scanning also looks for the laser pointer. The 3D corner points of the drawing surface, D_{1-4} (see Figure 3.1), can be converted to 2D points in the image frame using the camera model's conversion function, V:

$$d_i = V(D_i) \tag{4.1}$$

After the extrinsic parameters R and T are found, D_{1-4} can be converted to d_{1-4}. Then the red pixels that were found in the image scan can be tested to see if they fall within the region defined by the corner points. This is done to separate red laser spot pixels from red ellipse pixels. To test to see if a 2D screen point (x_p, y_p) falls within a polygonal region defined by d_i, consider a horizontal ray emanating from (x_p, y_p) and going to the right. If the number of times this ray intersects the line segments making up the polygon is equal to one, then the point is inside the polygon. Otherwise, if the number of intersections is greater than one, then the point (x_p, y_p) lies outside the polygon.

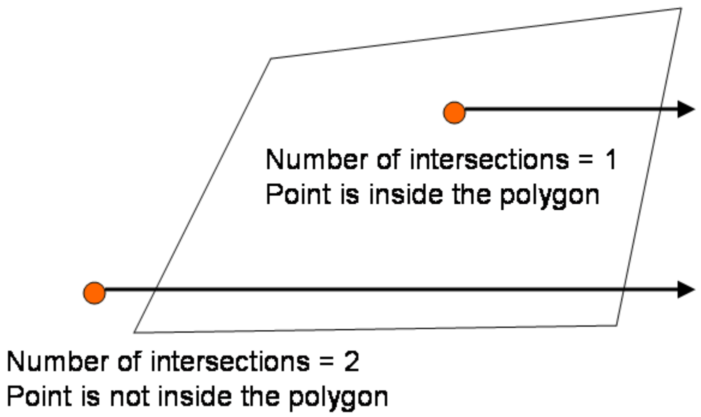
Number of intersections = 1
Point is inside the polygon
Number of intersections = 2
Point is not inside the polygon

Figure 4.1: Convex polygon test.

If a red pixel passes this polygon test, then it is stored. After all of the red pixels have been tested, the centroid, p_{pen}, of the stored pixels is calculated. p_{pen} is the location of the laser pointer in the Image Frame.

4.2 Calculating the 3D drawing location

Once p_{pen} is found, we calculate the corresponding 3D point in the World Frame. p_{pen} is converted from a point in the Image Frame to a point, P_{3D}, by applying the inverse of the camera model function, V. The vector L points from the location of the camera in the World Frame to P_{3D} (see Figure 4.2). The equation defining P_{3D} is:

$$P_{3D} = E + tL \tag{4.2}$$

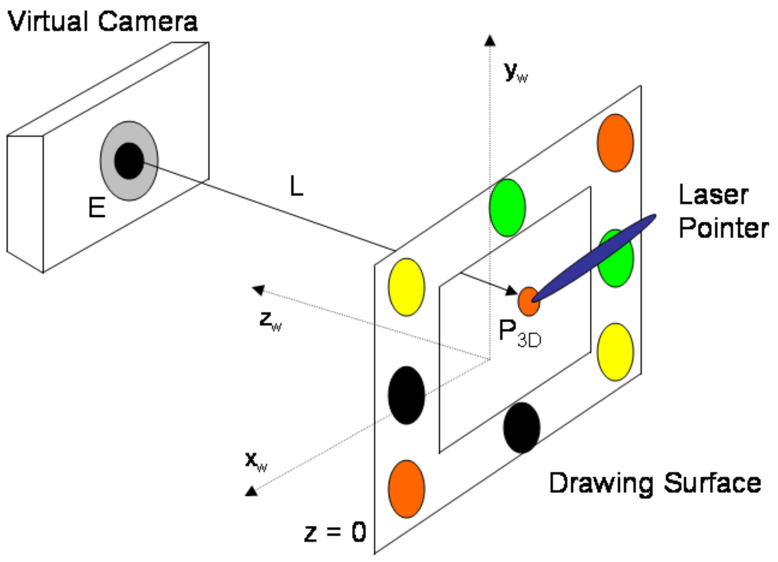

Figure 4.2: Tracking the laser pointer in 3D space.

where E is the location of the virtual camera in the World Frame. The value of E is obtained from the camera model. Because the drawing surface is sitting at $z = 0$ in the World Frame, the following is true:

$$E_z + tL_z = 0 \tag{4.3}$$

Solving for t:

$$t = -E_z/L_z \tag{4.4}$$

The equation for the 3D laser point in the World Frame is therefore:

$$P_{3D} = E + (-E_z/L_z)L \tag{4.5}$$

This 3D point, P_{3D}, corresponds to the point when the drawing surface is at $z = 0$ in the World Frame. The point needs to be transformed to the current location of the drawing surface in the World Frame. To do this, we multiply P_{3D} by the transformation matrix, $[R^T|T_d]$, that we found in Section 3.5. Now we can track the laser pointer as it moves in the World Frame.

Chapter 5

Sample Drawing Application

This chapter provides a description of a drawing application that uses the 3D input device. The drawing application allows the user to make characters by sketching ellipses. The 3D input device lets the user create these characters in a more intuitive manner. The drawing application creates a virtual rendering of the drawing surface and displays this to the user. As the user positions the real drawing surface, the virtual one mirrors its movements. A laser pointer is used to sketch out ellipse shapes on the drawing surface. The application uses the 3D data from the input device to create an ellipse shape in the virtual scene. The application consists of two virtual display windows that provide the user with different views of the virtual scene.

5.1 Virtual Display Windows

The virtual display consists of two windows that are displayed on the computer monitor. The first window is a re-creation of the 2D drawing surface (see Figure 5.1). The user can also see the cross-section of any shapes that the drawing surface is currently intersecting. While the user is drawing with the laser pointer, a trace of the pointer's movement is rendered on the virtual surface with a series of yellow dots.

The other window shows the 3D virtual scene. In this window, the user can see the virtual drawing surface's orientation in 3D space relative to the world. The virtual drawing surface's position and orientation in the world frame is calculated by applying the transformation matrix, $[R^{-1}|T_d]$, to the virtual drawing surface's starting position (see Section 3.5). Three virtual walls are also rendered to provide a sense of depth (see Figure 5.2).

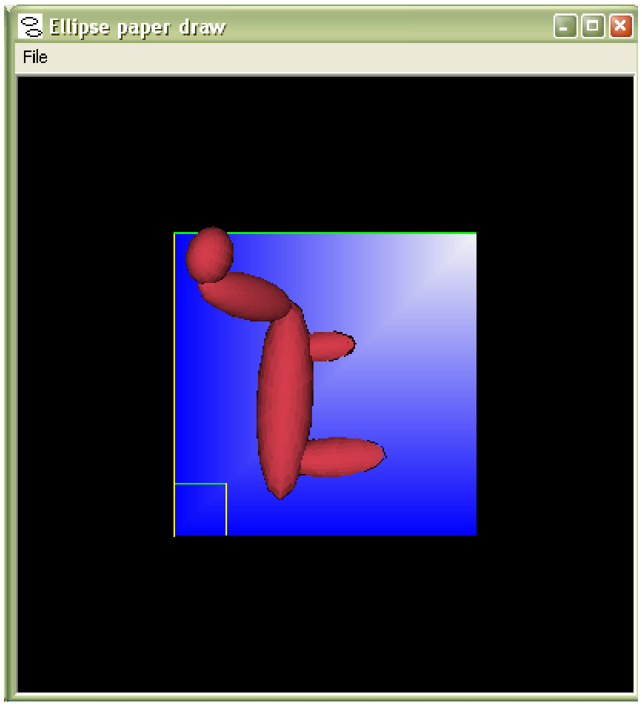

Figure 5.1: The virtual drawing surface window.

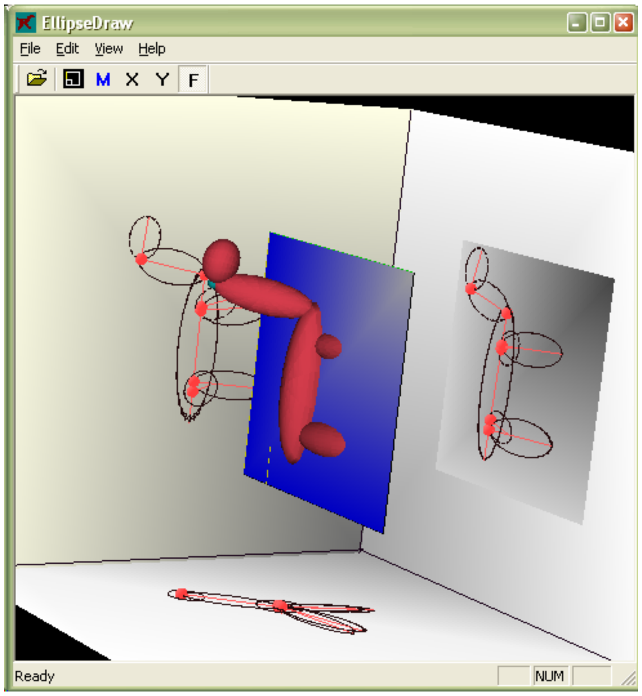

Figure 5.2: The virtual scene window.

Table 5.1: 3D Input Device Events to Mouse Events

Mouse Event	**3D Input Device Event**
Mouse Button Down	When the laser pointer is turned on for the first time.
Mouse Move	When the laser pointer is on and the location changes.
Mouse Button Up	When the laster pointer is turned off after being on.

5.2 Character Creation

Because this application does not use a mouse, the typical mouse events have to be mapped to the actions of the 3D Input Device. Table 5.1 lists the mapping of the typical mouse events to 3D Input Device events.

The drawing application receives the 3D points from the 3D input device as *Mouse Move* events. These points indicate where the laser pointer is at in the World Frame (see Section 4.2). The drawing application renders these points as small dots that appear in the window with the virtual drawing surface. These points provide feedback that lets the user know what kind of shape is being drawn.

When the application receives the *Mouse Button Up* event from the input device, it fits an ellipsoid to the 3D points. This ellipsoid is then rendered in the virtual scene. The user can join ellipsoids to create 3D characters. The characters can also be saved to a file and opened later for editing.

5.3 Pie Menus

Pie menus [12] are used in the drawing application to allow the user to change program settings and perform certain actions (see Figure 5.3). Each piece of the pie is a single menu action. Some actions bring up sub-menus that appear on top of the parent menu. The sub-menu is simply another pie menu that appears on top of the parent menu. The pie menu is brought up by shining the laser pointer in the lower left corner of the drawing region. When a menu item is selected, the pie menu is taken off the screen.

The user can select between the different menu items by turning on the laser pointer and moving it to the location of that item in the pie menu. While the user is moving the active laser pointer around, the 3D Input Device will be sending *Mouse Move* events to the drawing application. This allows the drawing application to determine which menu item the laser pointer is currently located on. The currently

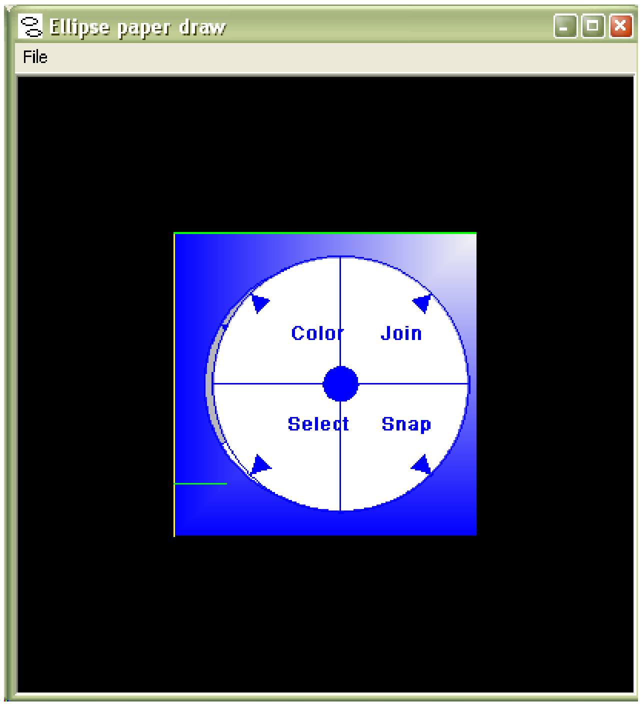

Figure 5.3: Pie menus.

selected menu item will be highlighted. When the desired menu item is highlighted, the user turns off the laser pointer. This triggers the *Mouse Button Up* event in the drawing application. The selected menu item is then executed.

The pie menus work well with the 3D input device. After continued use, the user grows accustomed to the location of certain menu items. Actions can be completed quickly with little input device movement required by the user.

Chapter 6

Results

6.1 Accuracy

The accuracy of the image scanning, ellipse fitting, and camera parameter searching was tested using a simulated version of the calibration pattern. A virtual camera is positioned at different orientations to view the display surface. The actual values for T and R are known in this case. The camera takes a snapshot of the scene and the ellipse search is performed. The starting point for the search is always $T = (0, 0, 5), R = (0, 0, 0)$. The tests were run on a set of 1,000 camera positions.

We wanted another case to compare our results to. So we took the known points on the ellipse borders in the World Frame and converted them to points in the Image Frame using the known values of T and R. This gives us points that lie directly on the ellipse borders in the image. We then used these points and performed the ellipse fitting and camera parameter search steps to get values for T and R.

Now for each of the 1,000 camera positions we have two sets of converged values for T and R. One is from performing the full ellipse search calibration method. The other set is from using the known ellipse border points instead of finding them in the image scanning steps. This creates the correct projected ellipses and removes error due to ellipse fitting.

We compared the converged values of T and R with the actual values of T and R. The values for the components of T are normalized to the range [-1 to 1]. R is broken down into three rotation angles around the x, y, and z axes. We also evaluated the error function, $E(T, R)$, using the two sets of values for T and R.

The following is a series of graphs that shows the results from this test. Figures 6.1 and 6.2 show the average difference between the converged values of T and

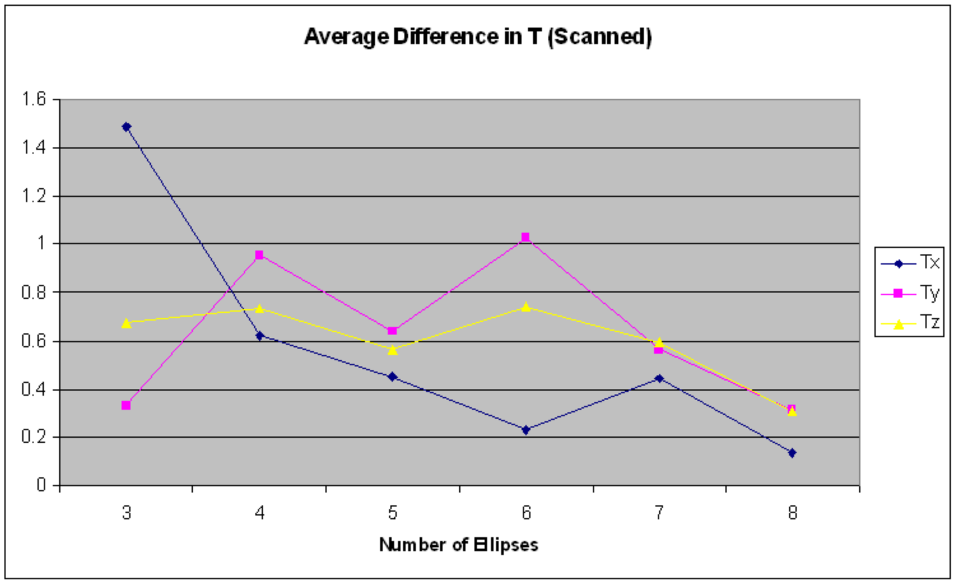

Figure 6.1: Results from Accuracy Test with the Scanned Points

the actual value of T. The average error is shown for the x, y, and z components of T. The data is broken down to show the average error per the number of ellipses that were detected in the image for that camera position. Figures 6.3 and 6.4 show the average difference between the converged values of R and the known value of R in degrees. That data for these two figures is broken down in the same manner as the first two figures. The last graph, Figure 6.5, shows the results when evaluating $E(T, R)$ using both sets of converged values for T and R.

One can see from the graphs that the average error gradually decreases as the number of ellipses increases. This would be expected since an increase in the number of ellipses provides the calibration method with more data to use in the minimization of the error function. It is also interesting to note in the last graph that the evaluated value of $E(T, R)$ goes down to almost zero for the known border points. It remains almost constant for the scanned points. One could infer here that the Simplex Search just reaches the minimum tolerance value for the scanned points, while the known points provide better results. This too would also be expected since we are using the actual ellipse border points from the Image Frame which should provide more accurate results.

Another point worth mentioning is that the Simplex Search finds a way to reach the tolerance value in some cases where there is a large error between the converged

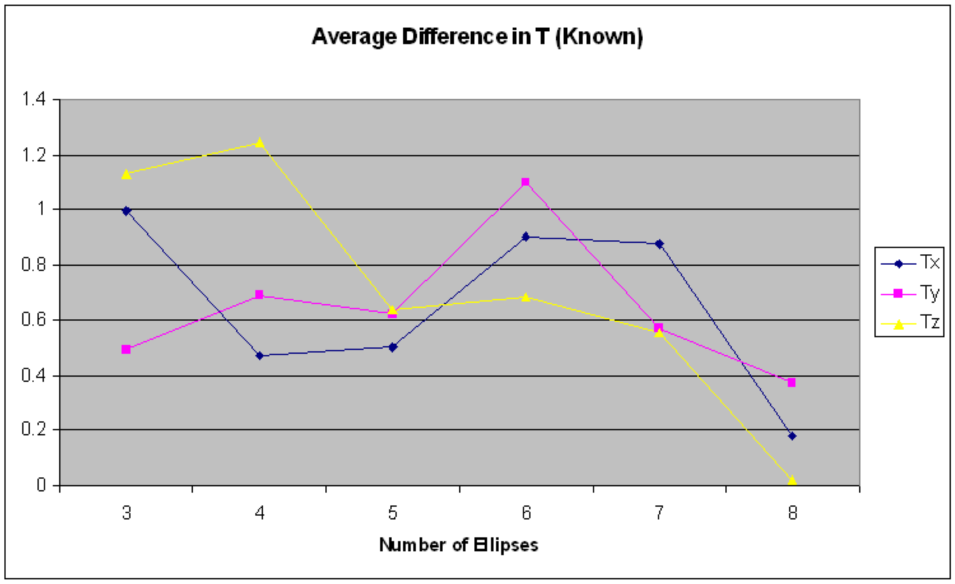

Figure 6.2: Results from Accuracy Test with the Known Points

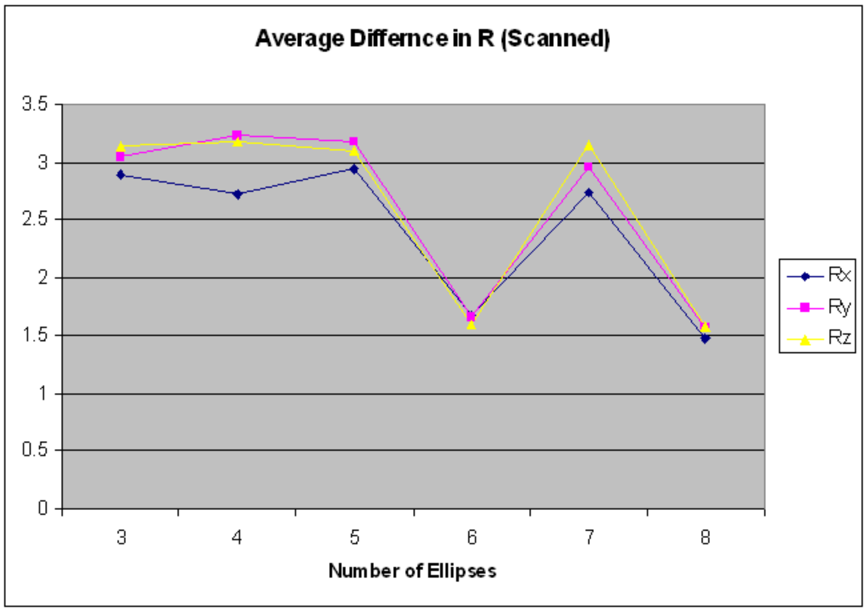

Figure 6.3: Results from Accuracy Test with the Scanned Points

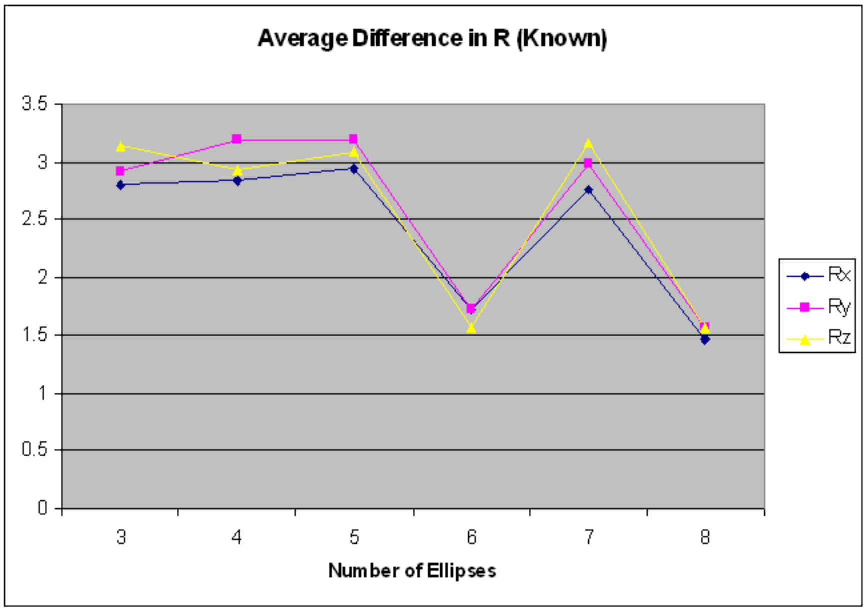

Figure 6.4: Results from Accuracy Test with the Known Points

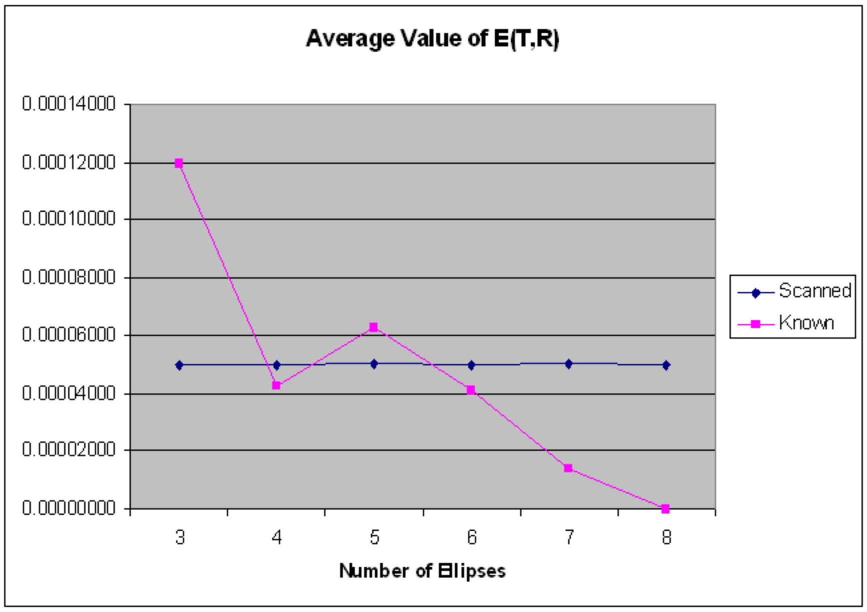

Figure 6.5: Results from Accuracy Test

and known values of T and R. This could be the case where a slightly different translation or rotation minimizes the error function as well as the correct values.

6.2 Real World Tracking

The accuracy of the ellipse search calibration method was compared to the OpenCV method [11]. A 5×7 checkerboard pattern was placed in the center of the calibration pattern. The ellipse search and the OpenCV calibration methods were then performed on a set of 10 images of size 640×480 pixels. The 10 images were snapshots from the digital camera of the drawing surface at different orientations (see Figure 6.6).

$E(T, R)$ was evaluated with the values of T and R found from both the ellipse search and the OpenCV method. The 3D coordinates of the 24 interior checkerboard corner points were measured. These points were then converted to 2D points in the Image Frame using $V()$ with the values of T and R from the ellipse search and the OpenCV methods. These 2D points were then compared to the 2D corner points that were found in a scan of the image. The average pixel distance between the calculated and scanned points was calculated for each image.

Table 6.1 provides a summary of the results from this test. The average value of $E(T, R)$ was lower for the ellipse search method than the OpenCV method. This is expected, since the ellipse search parameters are found by finding the minimum of $E(T, R)$. The average pixel distance for the checkerboard corners is lower for the OpenCV method than for the ellipse search method. One would expect the OpenCV results to be lower in this area because the OpenCV process is based on finding the interior checkerboard points. Another reason why the OpenCV results are better than the ellipse search is that the OpenCV method used the actual intrinsic parameters of the camera. We used our approximated guess for the intrinsic parameters in the ellipse search method. The overall performance of the 3D input device could be improved by incorporating a method for calculating the actual intrinsic parameters of the camera.

Figure 6.6 shows four of the input images used in the test. The green dots on the image represent the corner points that were converted to 2D image points using the ellipse search values for T and R. The red dots are the points that were converted using the OpenCV values for T and R.

Table 6.1: Real World Tracking Results

	Avg	**Min**	**Max**	**Std Dev**
$E(T, R)$ with Ellipse Search	0.0000104	0.00000976	0.0000111	0.000009742
$E(T, R)$ with OpenCV	0.0000531	0.0000455	0.0000606	0.0000107
Pixel Error with Ellipse Search	8.192	6.697	9.6855	2.112
Pixel Error with OpenCV	0.6007	0.5996	0.6017	0.00159

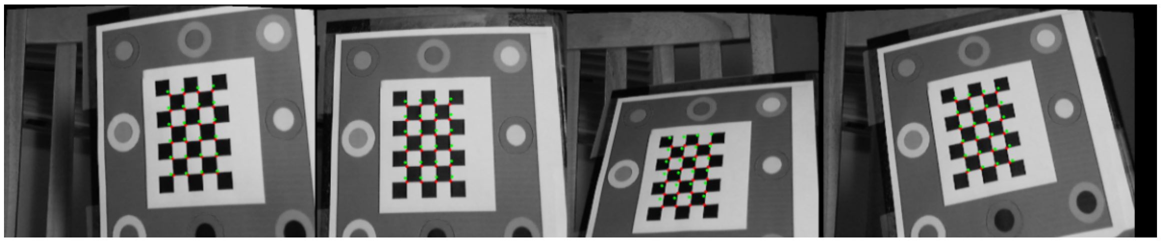

Figure 6.6: Real World Tracking Test Images.

6.3 Performance

The current system runs at approximately 10 fps on a 356MHZ Pentium, using $120 \times$ 160-sized images. The current bottleneck is the image scanning routine; hence the small image sizes. We filter the positions and orientation of the plane by taking the average of values from the previous two frames. This helps to reduce some of the noise in the converged values.

Chapter 7

Future Work

7.1 Performance Improvements

Improving the performance of the image scanning algorithm would greatly improve the current frame-rate. The linear search that we are currently using is not efficient.

A history of the border points could be kept and used as starting points for the next iteration of the image scanning. This would allow the image scanning software to find the border points without having to scan the entire image.

Enhancements in the computational performance of the image scanning algorithms would allow for the use of a larger resolution image. The current image being used is 120×160 pixels. The motivation behind such a small image size was to minimize the number of pixels that had to be scanned.

As mentioned in Chapter 6, the use of the actual intrinsic parameters should improve the performance of the input device. A method for calculating this parameters using standard techniques could be incorporated into the device. Doing this should improve the overall accuracy.

7.2 Projector Use

Another enhancement would be to use a projector to display the image that is seen in the virtual drawing surface window onto the actual drawing surface. The projected image would be warped to map it correctly to the orientation of the drawing surface. This would free the user from having to look at the computer monitor while drawing shapes.

With this enhancement, the application could also be used for analyzing 3D data sets. The user could simply move the display surface through 3D space and see a cross section of the data set that gets projected onto the display surface.

7.3 Tablet PC

A new line of laptops called the “Tablet PC” are being developed [16]. This new device resembles a regular laptop. The main difference is that the user can draw on the LCD display with a stylus-type input device. The LCD display can be rotated around so that it is facing up when the laptop is closed. A Tablet PC could be used in this application as the drawing surface. The calibration pattern would be put on underside of the Tablet PC. This would allow the user to orient the PC in front of the camera. The LCD display would then show the virtual display scene. This would allow the user to directly interact with the virtual display. The orientation of the laptop would determine the orientation of the virtual piece of paper. A laser pointer would no longer be needed as an input device. This would be a more accurate way of drawing shapes. Although, the weight of the laptop could make it difficult to use as an input device.

References

[1] Ronald Azuma. A Survey of Augmented Reality. *Presence*, 6(4):355–385, 1997.

[2] Jose Miguel Buenaposada and Luis Baumela. Real-time tracking and estimation of plane pose. In *ICPR*, pages 697–700, 2002.

[3] F. Dellaert, C. Thorpe, and S. Thrun. Super-resolved texture tracking of planar surface patches, 1998.

[4] Andrew W. Fitzgibbon, Maurizio Pilu, and Robert B. Fisher. Direct Least Square Fitting of Ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):476–480, 1999.

[5] George W Ftizmaurice and William Buxton. An Empirical Evaluation of Graspable User Interfaces: Towards specialized, space-multiplexed input. In *Computer Human Interaction*, pages 43–50, 1997.

[6] George W Ftizmaurice, Hiroshi Ishii, and William Buxton. Bricks: Laying the Foundations for Graspable User Interfaces. In *Computer Human Interaction*, pages 1–8, 1995.

[7] Beverly L Harrison, Kenneth P Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze Me, Hold Me, Tilt Me! An Exploration of Manipulative User Interfaces. In *Computer Human Interaction*, pages 17–24, 1998.

[8] Ken Hinckley, Randy Pausch, John C Goble, and Neal F Kassell. Passive Real-World Interface Props for Neurosurgical Visualization. In *Computer Human Interaction*, pages 452–458, 1994.

[9] H. Ishii and B. Ullmer. Tangible Bits: Towards Seamless Interfaces Between People, Bits, and Atoms. In *Computer Human Interaction*, pages 234–241, 1997.

[10] Q. JI, M. COSTA, R. HARALICK, and L. SHAPIRO. An integrated linear technique for pose estimation from different features, 1999.

[11] OpenCV: http://www.intel.com/research/mrl/research/opencv/.

[12] Pie Menus: http://www.piemenus.com/.

[13] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Downhill Simplex Method in Multidimensions. In *Numerical Recieps in C*, pages 408–412, 1992.

[14] Polhemus: http://www.polhemus.com/home.htm.

[15] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking. *Computer Graphics*, 30(Annual Conference Series):429–438, 1996. http://www.hpl.hp.com/personal/Bruce_Culbertson/ibr98/chromagl.htm.

[16] Tablet PC: http://www.microsoft.com/windowsxp/tabletpc/default.asp.

[17] Zhengyou Zhang. Flexible Camera Calibration by Viewing a Plane from Unknown Orientations. In *ICCV*, pages 666–673, 1999.

[18] Zhengyou Zhang, Ying Wu, Ying Shan, and Steven Shafer. Visual Panel: Virtual Mouse, Keyboard and 3D Controller with an Ordinary Piece of Paper. In *Perceptive User Intefaces*, 2001.

Vita

Mark A. Schroering

Date of Birth	October 15, 1976
Place of Birth	Jasper, Indiana
Degrees	B.S. Magna Cum Laude, Computer Science, May 1999

May 2003