Spring 5-11-2011

Implementation Of An Inexpensive 3d Scanner

Robert Michael Sivley
Western Kentucky University, robert.sivley354@topper.wku.edu

Recommended Citation

Sivley, Robert Michael, "Implementation Of An Inexpensive 3d Scanner" (2011). *Honors College Capstone Experience/Thesis Projects.* Paper 326.
http://digitalcommons.wku.edu/stu_hon_theses/326

IMPLEMENTATION OF AN INEXPENSIVE 3D SCANNER

A Capstone Experience / Thesis Project

Presented in Partial Fulfillment of the Requirements for

the Degree Bachelor of Science with

Honors College Graduate Distinction at Western Kentucky University

By

R. Michael Sivley

*****

Western Kentucky University

2011

CE/T Committee:

Professor James Gary (Advisor)

Professor Walter Collett

Professor Uta Ziegler

Approved by:

Advisor
Department of Mathematics
and Computer Science

ABSTRACT

The purpose of this project was to make 3D scanning technology accessible to the general public at an affordable price. To accomplish this, a scanner was designed that utilized structured-light projection. Coupled with triangulation, this provided an inexpensive, image-based modeling technique. Software was developed to analyze the required images and generate a 3D model. Users can edit analysis parameters during runtime to better optimize results for specific images. The 3D models generated are stored according to the .obj standard and can be opened in any commercial 3D modeling software. Results were positive, but several issues exist with the chosen scanning method; mostly pertaining to image processing. Applications are limited, but results suggest the scanner could be used, with modifications, for low-resolution scans at minimal cost.

Keywords: inexpensive, 3D, scanner, IBMR, gaming, webcam

ACKNOWLEDGEMENTS

Funding for this project came from grants provided by Western Kentucky University's Student Government Association and Honors College. Without their financial support, this research would have remained entirely theoretical. Thanks also to John Miller for his help in constructing the camera and target object stands, as well as his advice on lighting conditions. Lastly, thanks to my cousin, Robby Sivley, for the use of his computer during the early stages of the development process. There would have been several lost work days if not for his generosity.

VITA

November 16, 1988...Born – Bowling Green, Kentucky

2007..Greenwood High School, Bowling Green, Kentucky

2007..Western Kentucky University Award of Excellence

2010..Kentucky Honors Roundtable Presentation

2010..Barry M. Goldwater Scholar Honorable Mention

2010..NASA Langley Aerospace Research Summer Scholars (LARSS)

FIELDS OF STUDY

Major Field: Computer Science

Minor Fields: Mathematics

TABLE OF CONTENTS

Page

Chapters:

Appendices:

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

3D scanning is a technology that exists in many forms. Techniques for acquiring a 3D model from the environment vary from laser rangefinders to modulated light. While each mapping process has its costs and benefits, most implementations involve expensive equipment. Scanning equipment typically includes software packages, precision lasers, or high definition cameras and projectors. This tends to keep 3D scanning technology out of the hands of the average consumer. Some popular scanning solutions include 4DDynamics [0], NextEngine [1], and the David LaserScanner [2]. The David product sits apart from the rest in that it also attempts to achieve a low-cost solution to scanning. The system uses a laser line to highlight the curvature of a target object. The laser is swept across the object and data is acquired through real-time video analysis. David successfully created a system that functions with inexpensive equipment; however, the high-resolution scans showcased on their website are not products of cheap webcams and handheld lasers, but high-end, mechanized equipment.

The objective of this project was to develop a 3D scanner that would not only function with cheap equipment, but produce models of the desired quality with that same equipment. The project was inspired by independent game development studios, with a focus on those found at the college level. These studios tend to be understaffed, and any acceleration to the development process is valued. For those studios developing for the modern market, significant time is spent modeling various 3D characters. An inexpensive 3D scanner would allow studios to scan their own developers for initial models. The models would then only require minor adjustments.

The applications of this technology extend far beyond game development. Some popular applications of 3D scanning technology include facial recognition, parts measurement, and digitizing sculptures and other structures. Systems already exist to perform these activities, and the goal of this research is not to replicate previous work. Instead, the aim of this research is explore inexpensive methods for 3D scanning so that these techniques might later be applied to new products. Inexpensive solutions would benefit both businesses and consumers, increasing the potential market and lowering the cost of development.

CHAPTER 2

APPROACH

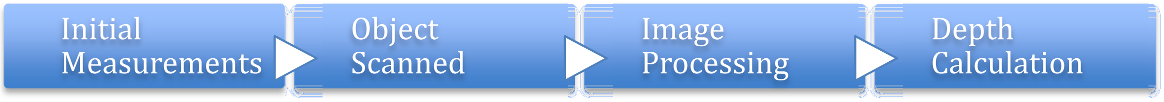
Equipment

This project used two Microsoft HD LifeCams ($100), a standard classroom projector, a flat, off-white surface, and separate wooden platforms for the projector and cameras ($10). At conception, the target audience for this technology was independent game developers at the college level, so the assumption was made that projectors would be available for use. However, if no projector is available, the cost effectiveness of this approach is reduced significantly. Towards the end of this project, the Microsoft Kinect [3] was released as a commercial solution to 3D mapping. For comparison, the retail price of a Kinect at the time of writing is $150.

General

Because the purpose of this research was to develop a scanner that could produce decent models while using inexpensive, low-quality equipment, a scanning technique had to be developed that produced precise models but did not rely on precise measurements. Laser scanning and range finders require expensive, specialized equipment. Likewise, techniques using modulated light or similar methods require cameras that can detect subtle changes in an image, a quality difficult to find in cheap webcams. In order to achieve decent resolution, it was necessary to abandon an attempt at high resolution and focus on data acquisition.

A combination of techniques was used, borrowing characteristics from structured-light projection, triangulation and stereoscopic measurement. A projector is used to display a 31x31 grid of blue points onto a target object, with a white reference point in the center. Each point is connected to its neighbors by green vertical lines and red horizontal lines[1]. The blue points in the projection map directly to vertices in the final model. This explicit declaration of vertices allows a low-resolution camera to still detect important features. The cameras are set equidistant from the background and centered on the grid's reference point.



---

[1] *see chapter 3 for all superscripts*

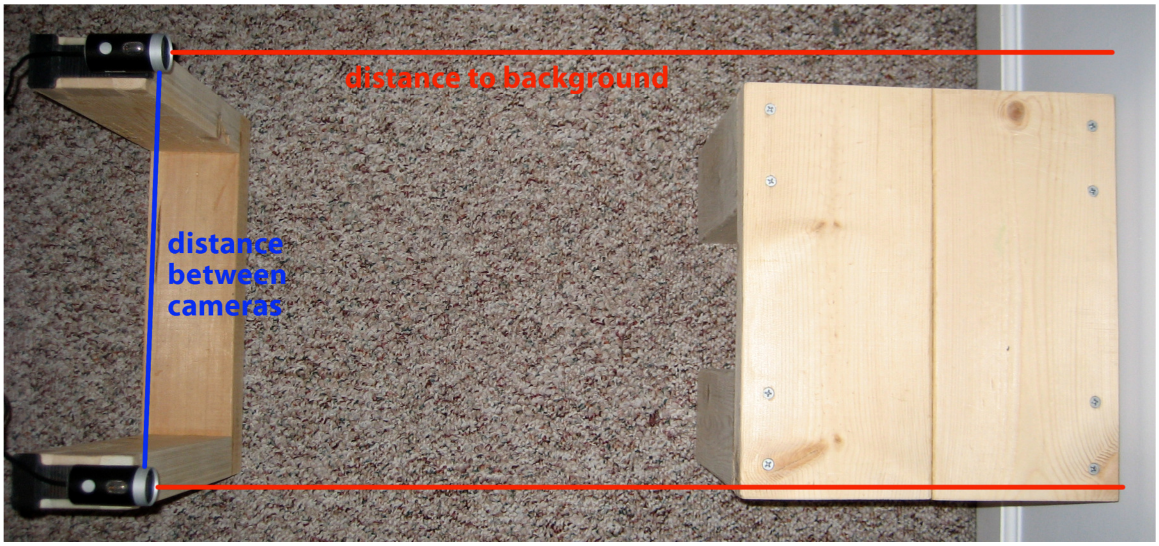


Fig 1. Overhead view of the scanner arrangement. A projector is centered behind the cameras and the grid is projected on the background to the right.

Initial photos and measurements are taken with the target object absent, and then scan photos are taken with the target object present. Using measured values and one approximate conversion, the distance to each point is calculated trigonometrically. This process will be described in more detail in the depth calculation and image processing sections. A 3D model is then generated as a grid of 31x31 vertices with the depth of each vertex set at its calculated distance from the cameras. Details for this process are described in the model construction section.

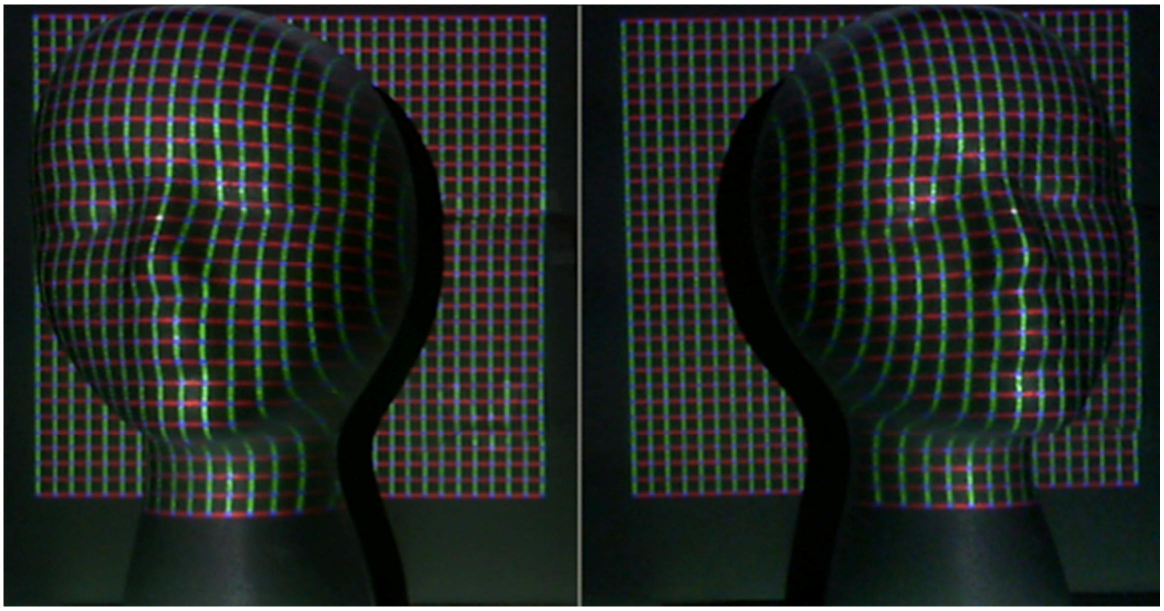


Fig 2. Cropped left and right photos of the target object with projection

Depth Calculation

This unique scanning method relies on stereoscopic imaging and trigonometry for depth calculation. Because the cameras are arranged parallel to the background, a perpendicular line from any point on the display to the plane of the cameras, along with the line from the camera to the point, forms a right triangle. This geometric property is used to solve for the distance from the point to the camera plane using known or approximated values.

The center vertex in the display (Fig 3: A) is used as a reference to synchronize the two images. The cameras are positioned at an equal distance from the reference vertex and set at an equal distance on either side of the reference vertex. This arrangement creates an isosceles triangle with vertices at each camera and the reference point. Bisecting the angle at the reference vertex creates two right triangles that allow for trigonometric operations. The display is projected against a flat surface and the distance from the background to the camera plane (*dist_to_bg*) is measured (Fig 3: B). The distance between the cameras (*cam_dist*) is also recorded. The angles at each camera (Fig 3: θ) are then computed from the measured distances using inverse tangent.

$$\Theta_L = \Theta_R = \tan^{-1} \cdot 2(\frac{dist\_to\_bg}{cam\_dist})$$

These known values are then used to approximate a general conversion variable, called the plate scale, which is used once the object has been placed in front of the cameras. The plate scale relates the horizontal distance between the center of an image and the vertex to the angular measure from that camera to the vertex.

$$plate\_scale = \frac{90 - \Theta_{L/R}}{dist\_in\_pixels}$$

Once initial measurements have been taken and the plate scale calculated, the target object is placed in front of the background. The reference point should be centered in the desired scan area and positioned away from any significant protrusions or depressions in the object. Photos are taken from each camera and for each observed point on the grid. The plate scale then converts the horizontal distance in pixels (Fig 3: c) observed between the center of an image (Fig 3: a) and a vertex on the grid (Fig 3: A) to the measure of the triangle angle corresponding to that grid vertex (Fig 3: α).

$$\alpha_{L/R} = 90 - pixel\_dist \cdot plate\_scale$$

Because the measured points no longer lie directly in between the two cameras, the length of each triangle base (Fig 3: C, D) must be recalculated before solving for the distance.

$$base_L = \frac{cam\_dist}{1 + \frac{\tan \alpha_R}{\tan \alpha_L}}$$

$$base_R = cam\_dist - Base_L$$

Using the approximated angles and relevant base lengths, the distance (Fig 3: E) is calculated from the plane of the cameras to the point. This value is calculated for both triangles and then averaged to compensate for any inaccuracy from previous approximations.

$$vertex\_dist = \frac{base_L \tan \Theta_L + base_R \tan \Theta_R}{2}$$

This distance is then stored as the depth component of each vertex's spatial coordinates.

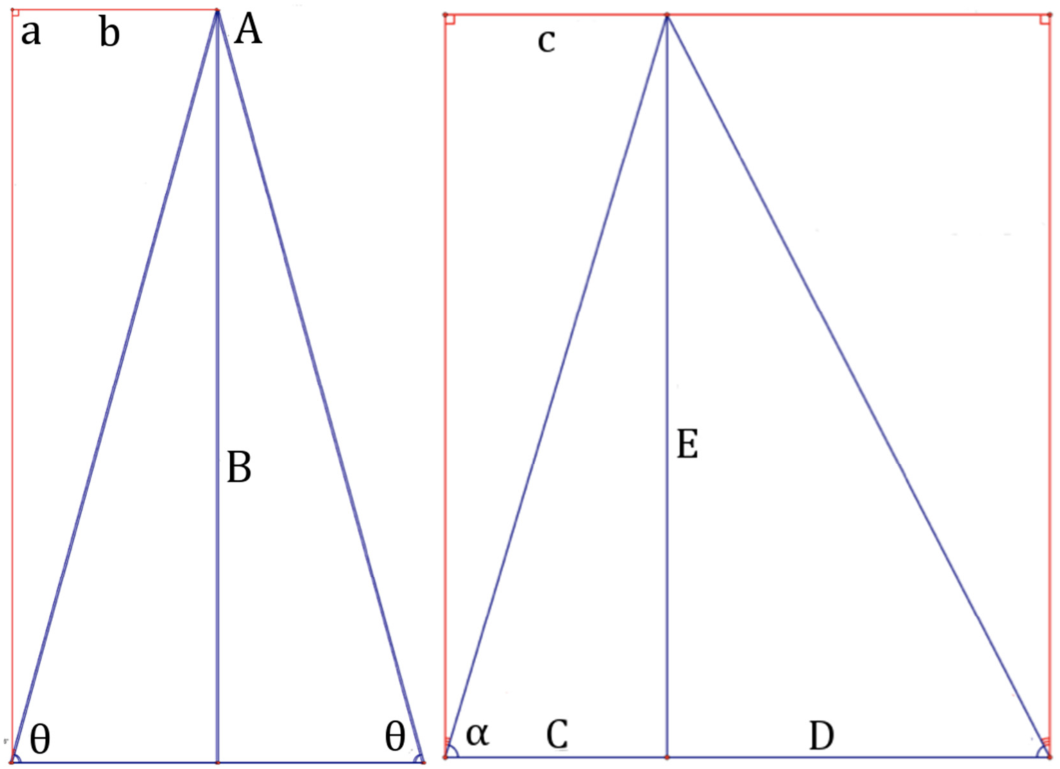


Fig 3. Diagrams of the initial measurements (left) and vertex distance calculation (right). Blue lines represent physical measurements (cm), red lines represent image measurements (px).

Image Processing

Once the initial measurements and the scan photos are taken, image processing is required to automate the scan from image to model. The data required from each image is the observed location of each vertex, and those vertices' actual position in the projected grid. Once this information is known, depth calculation can proceed. Computations include the identification of vertices and edges in the image, the linking of adjacent vertices using observed edges, the assignment of observed vertices to their actual position in the grid, and the estimation of obscured vertices. Before an image can be processed, the user must manually edit the photo in two important ways. First, everything in the image besides the display must be set to black to ensure that no erroneous data is read from surrounding areas. The less ambient light present in the image, the less important this adjustment becomes. Secondly, the reference point is a crucial component of image processing, and any error in identifying it derails the program immediately. The user must add a purely white dot over the observed vertex and ensure that no other purely white pixels exist in the image. The latter requirement is usually not an issue, but should be acknowledged none the less. These edits can be performed in Photoshop, Gimp, or any other photo editing software. Automated preprocessing of the image reduces the colors of all pixels to black, red, green, blue, or white. This determination has a default range, but can be adjusted by the user during runtime to accommodate lighting conditions[2]. Adjacent pixels with the same color are then clustered together. Each instance of clustering is run in parallel. Based on the color of its pixels, each cluster is converted into a vertex, horizontal edge, or vertical edge classes. Vertex classes store information about their location, color, connecting edges, and actual position in the grid. Edge classes are classified as either horizontal or vertical based on their color. Red edges are classified as horizontal and the locations of endpoints are calculated using the horizontal extremes of the pixels in the cluster. Green edges are classified as vertical, and their endpoints are calculated using their vertical extremes. Edge classes also store information about their color and those vertices to which they are connected. Edges with endpoints in close proximity to a vertex are then assigned to that vertex, and the connection is stored in both the vertex object and the edge object. The result is a vertex net[3], which functions as an undirected graph. The vertex net is then traversed in parallel, beginning with the reference point, known to mark position (15,15), and working outwards along edges, assigning each vertex its true position in the grid in relation to the previously visited vertex. At this point, the user is presented with a preview of the results and a set of sliders (Fig. 4). These sliders allow the user to adjust the range of hues identified as particular colors, as well as the minimum allowable values for each color. The purpose of this addition is to allow imperfect images to be used for scanning. For example, if the object's color interferes with the observed color vertices or edges, a new hue range can be specified that is unique to that image. Likewise, if poor lighting conditions were used or the projector is subpar, the minimum or allowable value for each color can be lowered to accommodate the darker display. Once the vertices have their observed locations and actual positions stored, they are returned for depth processing.

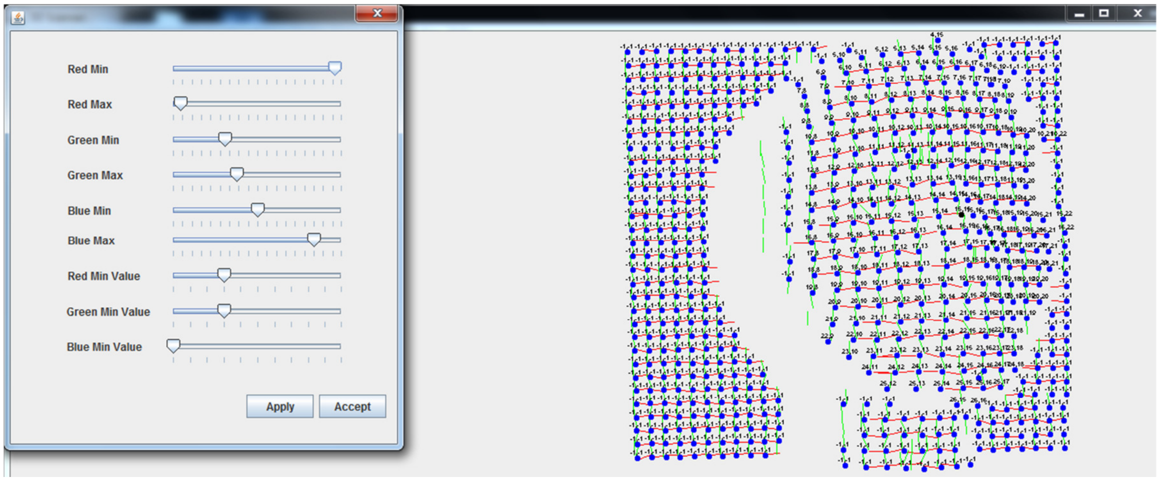


Fig 4. Sample screen during image processing. The preview on the right shows the observed vertices and the edges that connect them. Above each vertex is listed its calculated actual position in the grid. If errors exist, the option pane on the left allows the user to adjust for various lighting and color conditions.

3D Model Construction

Image processing and depth calculation are the most difficult elements of 3D scanning. Once the x, y and z values of each vertex have been determined, generating a simple mesh is fairly straightforward. According to the .obj standard [4], each vertex is listed, followed by the definition of each normal. The file can then be imported into any 3D modeling software for further editing[4]. If the user would prefer a point cloud rather than a mesh, the .obj file is easily human read. The second half of the definitions, those containing four values, must be deleted.

CHAPTER 3

RESULTS

To evaluate the success of this project, it is important to convey the obstacles that impeded development. Most of these issues were caused by the sacrifices made to keep costs low. Many were resolved, and the solutions may help in designing similar systems. However, there also some issues that have not been resolved, and would be good topics for future research in this area. Some statistical accuracy analysis is presented. Lastly, some sample models are included for reference.

Resolved Issues

[1]Originally, the projected grid used only two colors. The low-resolution cameras would record a glow around the edges, which would intersect the glow from nearby edges. During pixel clustering, these edges would be clustered together and evaluated as a single edge. By introducing a separate color for horizontal and vertical edges, the intersecting glow produces a composite color that is not clustered with either edge. Additionally, the reason that the hue range for blue covers a wider range than the others is to accommodate the purple shade created by intersecting red edges and blue vertices. Attributing these intersections to vertices instead of edges causes fewer collisions.

[2] The hue range and value limit for each color is hard coded based on ideal conditions. During testing, varying lighting conditions caused the images to shift along the hue spectrum, disrupting the static values. Colorful objects also caused these issues. To avoid requiring the user to achieve perfect conditions, the option was added to adjust these values during runtime. This change makes for a more flexible and user-friendly system.

[3] Originally, only the vertices were going to be displayed on the target object and the actual positions of observed vertices were to be determined by their position in relation to certain rows. The idea was that the correct row and column could be determined for a vertex by comparing its position to the edge vertices. However, when an object contained a significant protrusion or depression, the vertices shifted far enough such that assignments were incorrect. The edges were added to the projection to allow the program to traverse the vertices with increased confidence.

[4] The vertex net traversal was first designed as a parallel process, but written as sequential for testing. It was assumed this would only decrease efficiency. It was found that this method actually produced incorrect assignments. The cause for this error was the way the program would continue checking in one direction until it reached the edge of the grid. Towards the edge of an object, the observed vertices become less reliable. The program was accessing unreliable vertices too early, and the values of all other vertices were being determined in relation to the edge. By implementing parallelism, where each process

checks either the left, right, top, or bottom vertex in relation to the current, the reliable, interior vertices are assigned before unreliable data is reached.

Unresolved Issues

[5]The generated 3D model will consist of a 31x31 grid of vertices, including those vertices without enough information to calculate a reliable depth. All erroneous vertices will map to the same depth and must be deleted once opened in a 3D modeling program.

[6]In several of the image processing calculations, a variable *avg_separation* is used to approximate a reasonable distance between any two vertices. This value is calculated by selecting two adjacent vertices and using their horizontal separation as an average. This is a poor solution, and attempts have been made to phase the *avg_separation* value out of the program entirely. While alternative solutions have been implemented throughout most of the program, there are still some operations that rely on this value. The most notable problem caused by errors in this variable is the hyper-extension of edges. When a partial edge is recorded, its length is adjusted using the *avg_separation* value. This can usually be resolved by a minor color value adjustment. Below is a trivial example where the blue range was manipulated to the point that *avg_separation* was sure to be skewed.

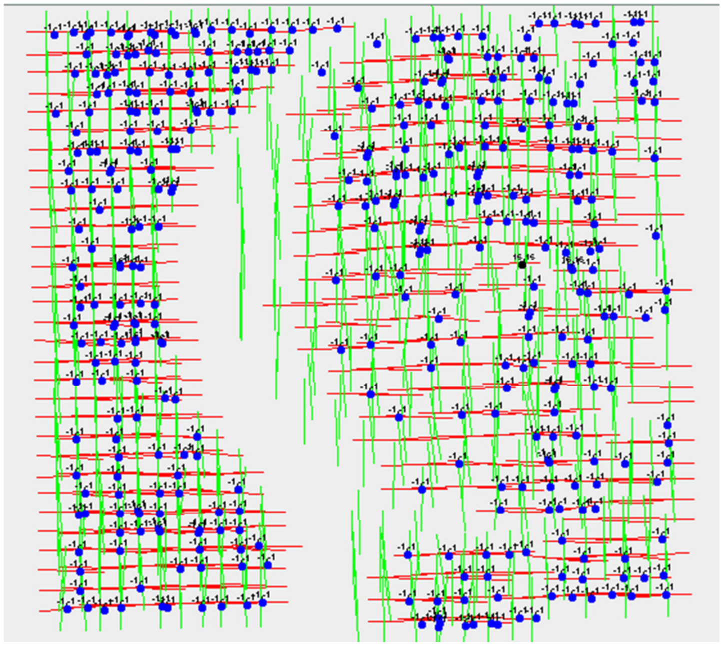


Fig 5. Trivial example of possible errors caused by unreliable avg_separation variable

[7]During the user input stage of processing, sliders are presented to the user to adjust the range of hues and minimum values allowed for each color. While the values chosen by the user are stored between selections, the sliders reset to their default positions each time they are refreshed. A solution has not been found and attempts to resolve the issue have caused the dialog box to fail.

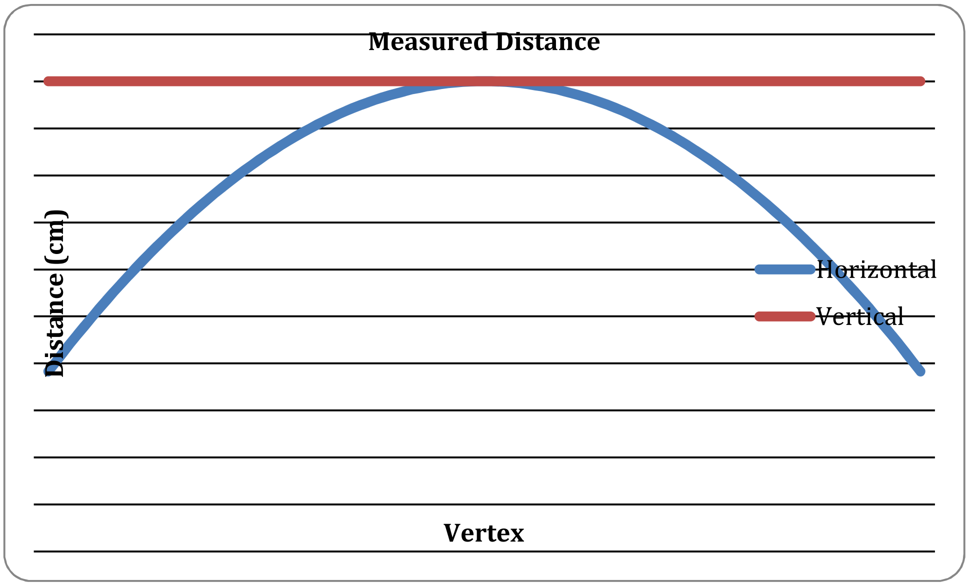
[8]If the user spends too much time adjusting color values, the program begins to lag and eventually crashes from an *OutOfMemory* error caused by heap space overflow. It is unclear whether the issue is caused by the amount of time spent adjusting values or the number of times the values are adjusted. It is possible that some data structures generated

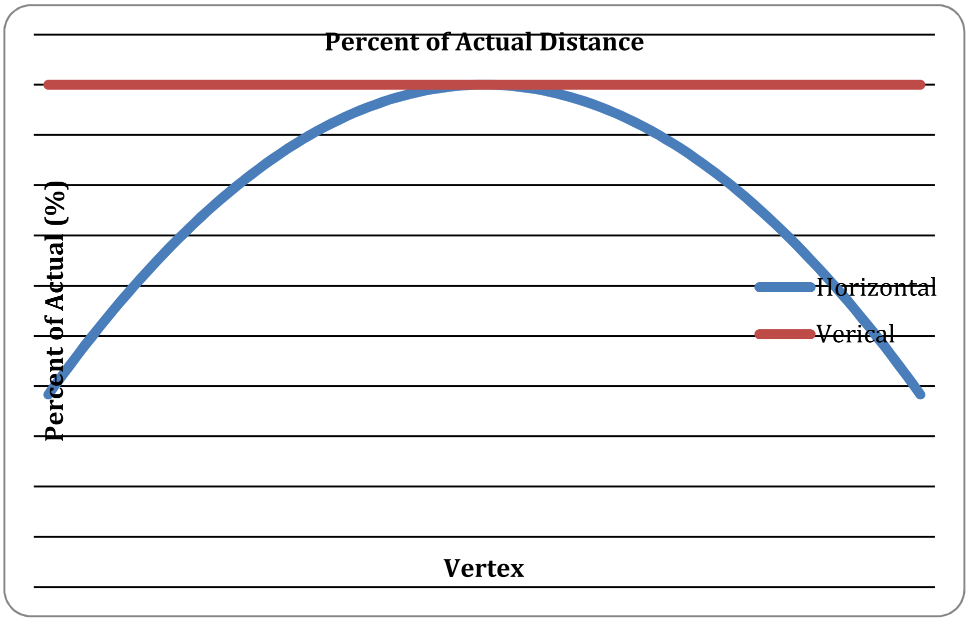
during the user input section are not being properly deallocated between iterations. However, the actual cause of this issue has not been identified.

[9] Pixel clustering and vertex net traversal are the two instances of parallel processing in the program, excluding the GUIs. In each of these, multiple threads pass over the same array of objects, mark the objects as having been evaluated, and then evaluate them. In neither process is a proper mutex implemented. It is possible that two threads could be created to process the same object if the first does not create a thread and mark the object before the second checks whether it is marked. It is believed that these collisions would only occur with frequency towards the beginning of the process, near the reference point. In this region, data is reliable and the data is likely overwritten with an equivalent value. Results have not shown any indication that an error is occurring because of this oversight. However, the fault is recognized and correct behavior cannot be guaranteed.

Error Analysis

The plate scale conversion is an estimate. To get an idea for the accuracy of these estimates, the background is scanned and calculated. Because a background scan places all vertices at the same distance from the cameras, errors can be found with high confidence. The diagrams below present the calculated error from three perspectives. The data indicates that, as one would expect, vertical deviation from the reference point does not affect accuracy, but horizontal deviation does.


Measured Distance
Distance (cm)
Horizontal
Vertical
Vertex


Percent of Actual Distance
Percent of Actual (%)
Horizontal
Verical
Vertex

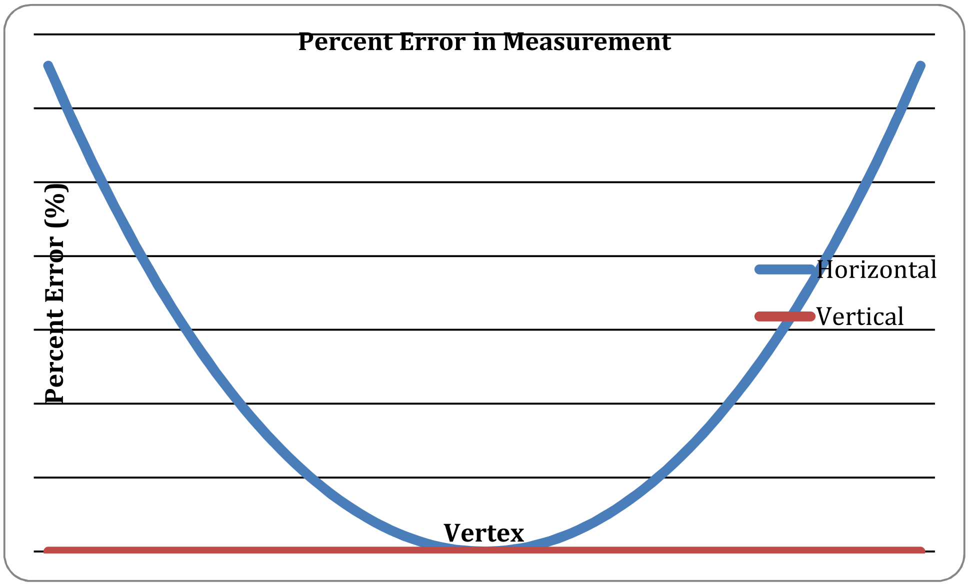


Fig 6. These graphs evaluate the same data from three different perspectives.

Models

The depth calculation portion of this project was a success. When the program is fed perfect input, bypassing any image processing, it generates a low-resolution mesh of sufficient quality for use in game modeling. While no calculations for statistical accuracy were performed, a qualitative analysis shows a very accurate depiction of the target object.

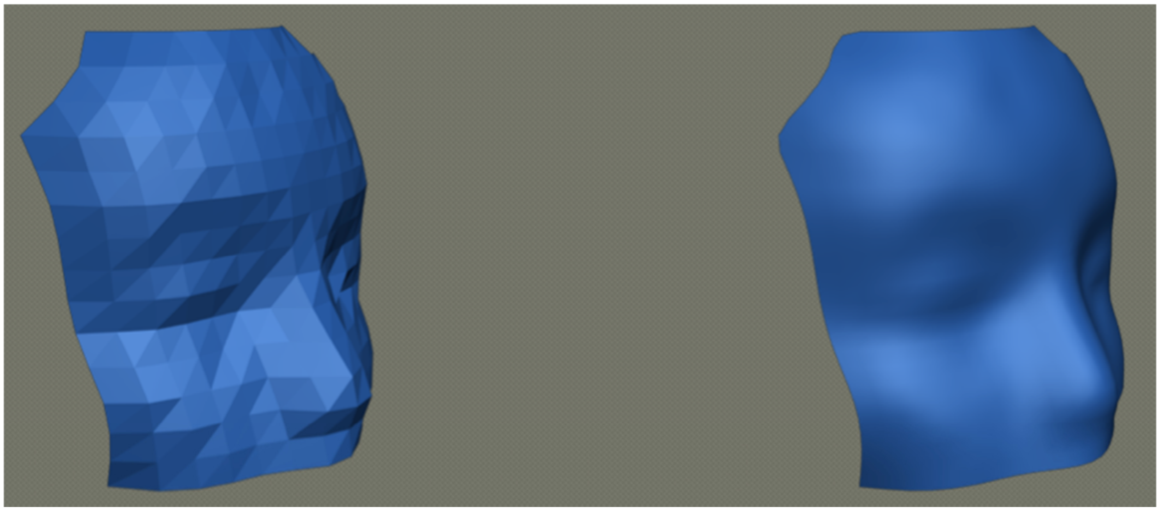


Fig 7. An example output model created by feeding perfect data directly to the depth calculation and mesh generation portions of the program. Data was obtained by hand measuring the x,y coordinates of each vertex in the both the left and right images.

The image processing portion of the project was much more challenging. Low quality equipment means noisy images, which make processing difficult. However, after several revisions to the image processing code and the addition of user input to customize the process for each image, the program now reliably outputs a partial scan. The scan is clearly of inferior quality to the previous example, but achieves its purpose. One aim of the project was to provide models of reasonable quality. This restriction is the reason for a partial scan when seemingly so much information is available. Only those observed points deemed reliable are included in the final model. With the use of third party software, partial scans from multiple viewing angles can be pieced together to form a complete 3D representation of an object, while maintaining the integrity of the scan.

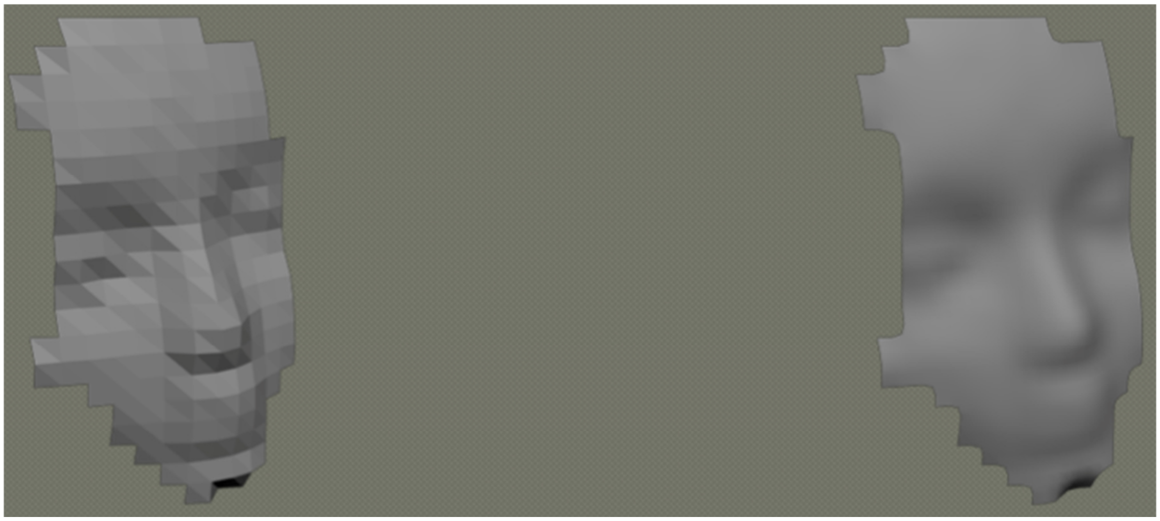


Fig 8. An example of a model created using only those facilities provided by the program.

The model includes only the region of the target object that can be reliably calculated.

CHAPTER 4

DISCUSSION

The inspiration for this project was independent game development. The goal was to develop an inexpensive system that could generate low-resolution models to decrease modeling time. While lacking in some areas, the project meets those expectations. The models generated by the program contain all of the major features found on the target object, and portrays them with reasonable accuracy. The model's resolution can be chosen by shrinking or expanding the surface over which the grid is projected, but current results fall within the allowable range for most game engines. The use of third party software allows for the partial scans created by the program to be merged into a complete model.

There is significant room to improve this project through future work. Currently, if a vertex is observed by one camera, and estimated by the other, the distance to that vertex is calculated like any other, using the values provided from both images. Results might be improved if the estimate is used only for base length calculations, while using only the reliable data to calculate the depth. The averaging that takes place is meant to accommodate any error made in estimation, but by using an estimated value as part of the average, it may be exacerbating the original problem.

A major issue concerning edge recognition is the glow that appears around the edges. This problem was significantly improved through the use of two different colors for vertical and horizontal edges, but there are still errors. A possible fix may be to analyze the object using two sets of photos. The first set would process as normal, while the second set would invert the grid, displaying the horizontal edges in green and the vertical edges in red. Then, only pixels that meet the color requirement in both sets of images would be considered valid. Because each color glows differently, incorrect pixels are unlikely to overlap, while correct pixels will. This improvement could allow for a wider range of values to be considered during processing, increasing the total number of pixels used in the assignment process.

Also, this system may translate well into a laser sweeping technique, where data is collected through real-time video analysis. This substitution would allow for significantly higher resolution scans if the resolution of the cameras allowed for enough precision. This might be implemented by using a vertical laser line and recording a single vertex for every row pixel in an image. The horizontal location would be calculated by averaging the position of all red pixels found on a row. Points could then be coordinated between images by synchronizing frames once either camera has detected any red. The vertex array could then be passed to the depth calculator of this program without any edits. An expected problem with this approach is the perfect alignment that would be required from the cameras to guarantee the same vertices are being processed in both images. Calibrating the images before scanning could determine an offset.

Finally, the release of Microsoft’s Kinect platform has encouraged a surge of 3D mapping applications. The Kinect offers a solid platform for stereoscopic vision as well as demonstrated success in 3D mapping. Substituting this product for the webcams may prove worth the slight cost increase. Further research would need to be done to determine if the additional sensors provided by the Kinect could eliminate the need for a projector. However, it is assumed that without the use of structured light projection, a much more robust feature recognition system would need to be implemented for image synchronization.

In its current state, this 3D scanner provides an affordable solution to a complex problem. The models it produces are of high enough quality for use in game development, and most modelers would be able to quickly correct for its shortcomings. With further improvements to the scanning technique and image processing, this project has the potential to become an accessible scanning tool. Most importantly, this research supports the premise that useful 3D scanning tools can be developed at minimal cost.

REFERENCES

[0] 4DDyanmics: Mephisto 3D Scanners. Antwerp, Belgium.

http://www.4ddynamics.com/. 04/07/2011.

[1] NextEngine: 3D Laser Scanner. Santa Monica, California.

http://www.nextengine.com/. 04/07/2011.

[2] David: LaserScanner, Shapefusion. Koblenz, Germany.

http://www.david-laserscanner.com/. 04/07/2011.

[3] Microsoft: Kinect. Redmond, Washington.

http://www.xbox.com/en-US/kinect. 04/25/2011.

[4] Wavefront .obj file. In *Wikipedia*. Retrieved April 25, 2011, from

http://en.wikipedia.org/wiki/Wavefront_.obj_file.