Building a 3D Laser Triangulation Scanner and 3D Printer

Ishani Thakur and Nikhar Arora

Dr. Dann
Applied Science Research - F
May 18, 2017

1.0 Abstract

For this project a functioning 3D printer and 3D scanner were constructed. A Prusa i3 V2 printer was built from a kit and programmed using an Arduino Mega. It was manually calibrated and leveled to print multiple objects such as a cube, a SF giants logo, and a ring that was previously scanned. The 3D scanner used a logitech webcam and a single red line laser to construct a point cloud of a given object. Once the point cloud was constructed, it was placed into a processing software, Blender, to remove extraneous points. The edited point cloud was imported into Meshlab, a software that computed normals and a Poisson surface for the object. This ultimately allowed the point cloud to convert to an STL file that was able to be printed, as was done with a red ring.

Table of Contents:

2.0 Big Idea

The big idea behind this project is to create a 3D scanner that uses laser triangulation methods to create 3D model of different objects. In addition, a RepRap 3D printer is being built and programmed using a DIY kit for testing of the 3D scanner.

There are many practical uses for 3D scanners that motivated this project. First, the most important use for 3D scanners is to create 3D printable models of objects that are difficult to quickly design on a computer. In addition, scanning can be a much faster process than using computer-aided design. 3D scanners in general are quite expensive; however, building one from scratch tremendously lowers the price, from approximately $300 to $75. There are many products that can be scanned using a 3D scanner, including museum artifacts, for educational uses and factory parts, that don't need to take up warehouse space, and instead, can be scanned and printed quickly as replacements. These are the ideas for use behind this 3D scanner project. The basic premise for this project is a turntable attached to a stepper motor that slowly rotates, while a Logitech webcam takes 2D images, and uses laser triangulation to convert the images into a 3D point cloud. Then, the point cloud is placed into MeshLab, an application that triangulates the point cloud points into a 3D representation. The 3D printer, which will be built from the RepRap kit, will take the files provided by MeshLab, and print the object.

The idea for this project came from 3D scanners that were read about in a Makezine article. In *5 DIY Scanners to Watch*, Patrick McCarthy describes laser-automated 3D scanners that have recently been developed. This includes Spinscan, a digital camera and laser-automated scanner and FabScan, that encloses the laser and camera system to prevent light distortion [1]. These up and coming scanners inspired the decision to build a laser triangulation 3D scanner. In addition, this project was chosen because there are many objects that cannot be accurately designed using CAD, but need to be 3D printed. In addition, the cost setback of buying a 3D scanner motivated the idea to build one.

There are many components of this adventure. In terms of the 3D scanner, the mechanics are critical in producing a usable scan. The lasers need to be placed at accurate angles and the turntable needs to be placed at a certain distance from the webcam and lasers. This project will allow for a greater understanding of mechanics to a greater extent throughout building the 3D scanner. There are also electrical components to the scanner. Each laser works using a transistor, Arduino, and power supply. Lastly, there is a large software component to the scanner. The first part of the programming is collecting a series of 2D images while the turntable slowly rotates and the lasers turn on and off. Next, the code needs to convert the 2D images into a 3D point cloud that can be placed in MeshLab to create the 3D representation of the object.
In terms of the 3D printer, the kit provides guided instructions on building the printer. The parts are provided; however, mechanical skills are needed to ensure success in building the printer. In addition, while programming guides are provided, the responsibilities for creating the code for the 3D printer is given to the customer.

All in all, there are many components that will be studied throughout this project. In addition to gained technical knowledge, skills of collaboration, time management, and organization will also be learned.

3.0 Introduction

The endeavor of building a 3D scanner and a 3D printer opens up possibilities in the fields of medicine, engineering, anthropology, and the media. The first most rudimentary form of 3D scanning was the Egyptians creating plaster casts of mummies' heads. This ability to capture contours and 3D surfaces was impressive, but time consuming and required multiple different materials such as linen and plaster. After the advent of computers, the early scanners used physical probes that touched the desired object thousands of times to create the digital model [2]. However, this took a lot of time and large amount of computing power.

3D scanning has since evolved into two different types, contact scanners and noncontact scanners. Noncontact scanners are of two types, time-of-flight and triangulation. Time-of-flight scanners use a laser rangefinder to evaluate contours, while triangulation scanners use lasers and a webcam sensor to probe the object [3]. The details on how a triangulation scanner functions is included in Section 4.1. A diagram of of a time of flight scanner is located below.

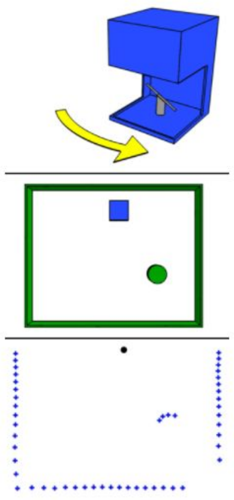


Figure 1: Diagram of Time of Flight laser scanner.

Essentially a laser is reflected off of a rotating mirror that hits points in the field of view. The laser is reflected back onto the mirror and into the rangefinder which measures the time it takes for the beam to make a round trip. The distance between the scanner and the object is calculated by using d = c*t/2, where t = the time it takes for the beam to take a round trip flight. This allows the scanner to capture the contours of one horizontal layer. The scanner is typically mounted on a set of rails that allows it to move in the z direction, allowing all points to be captured [4].

3D printers are also incredibly useful in these fields of engineering, medicine, entertainment. There are many types of 3D printers that exist today. The most common of these is FDM or fused deposition modeling, which is what our team will build. Essentially, objects are built layer by layer. The CAD model is sliced and then translated into a collection of x, y, z coordinates. The printer then heats thermoplastic to its melting point and extrudes the material onto a base.

FDM printers unfortunately produce objects with significant ribbing, in turn the FDM cannot print intricate objects. Furthermore, these printers are very slow to build large objects. An SLA printer, or stereolithography, works in a similar fashion to an FDM printer, building objects layer by layer. However, SLA printers use resin and after each layer is finished, SLA printers laser form the object. Furthermore, instead of the extruder moving along the z axis, the base itself moves along the z axis. To remedy the issue of holes in objects, support materials such as wax are often used during printing and then removed after completion. Unfortunately, SLA printed objects require post-curing and experience experience slight (<1%) shrinkage due to the phase change [5].

The other category of printers, SLS printers (selective laser sintering) and SLM printers (selective laser melting) uses powdered material instead of liquid material. Similar to an SLA printer, SLS printers print layer by layer. Between each layer, powdered material (nylon) is rolled over the previous layer, and is then melted together using a laser to form the object. SLM printers use a similar technique to SLS printers, but instead use a high powered laser to fuse together powdered metal instead of nylon. SLM printers are extremely advantageous because they can print in metal. SLS and SLM printers are both hard to use and produce objects with a rough finish. Furthermore, they are expensive due to high powered lasers and material changeover is difficult [6].

3D scanners and 3D printers combined are incredibly useful in multiple fields. In terms of healthcare, scanners can be used to gain an individual's anatomy. Printers can turn these scans into customized prosthetics and orthotic devices. Not only does this increase the device's effectiveness, but it also eliminates the need for universal devices that are accessible to multiple people. Scanners also aid with creating CADs of objects with contours that are difficult to measure, allowing rapid prototyping using a 3D printer. Furthermore, one can obtain the precise measurements of an object after scanning it. Another application of 3D scanners and 3D printers is anthropology. Multiple researchers have scanned bones and artifacts to preserve their original structure and create replicas to be distributed to other research facilities. Lastly, in terms of the media, video game and movie CGI is heavily constructed using 3D scanners. For example, the terminator suit was created by 3D scanning Arnold Schwarzenegger [7]. Furthermore, 3D printed materials have already been applied in many areas of the aerospace sector. It has the potential to make more significant advancements in the field. Many automotive industry companies are also looking into 3D printing for producing replacement and spare parts, rather than having a large holding inventory. Many artists have made names for themselves working with 3D printers and scanning technologies and creating art and sculptures. 3D printing can be used to produce accurate demonstration models' of an architect's vision. Also, some architects are looking into 3D printing as a direct construction method. Apis Cor recently built a 3D printed house in Russia. It took 24 hours to build the 4,000 square foot home at a cost of just above $10,000. The printer essentially functions in the same way as an FFF printer, laying down each layer of concrete and allowing the layers to fuse together. However, instead of the extruder moving in the x, y, and z directions, it moves in accordance with a cylindrical coordinate system. Essentially, the extruder is set at a certain height, moves to a certain radius, and deviates a set angle from the origin, allowing it to extrude material at all coordinates. [8] Some companies have also looked into using 3D printing to create sugar and chocolate creations. In addition, others have started experimenting with "meat" 3D printers [9].

4.0 Theory of the Project

4.1 Theory of 3D Scanner

The scanner being built is a triangulation scanner. Essentially, it uses laser light to capture the contours of an object.

First a line laser projects a line of light, highlighting a strip of an object, as depicted by Figure 2.

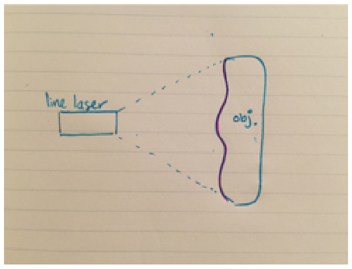


Figure 2: Line laser projecting light onto scanned object.

A laser is formed by exciting electrons in certain atoms, gases, or elements to move from a low energy orbit to a high energy orbit. These electrons become "excited" by absorbing energy from an electrical current. When the electrons move back to the lower energy level, photons are emitted, forming light that is all of the same wavelength and in phase. Therefore, it becomes a single beam [10]. A line is formed by diverging the beam of the laser itself with a Powell lens. A Powell lens takes this beam of light and splits it, creating a line, as shown in the below figure [11].

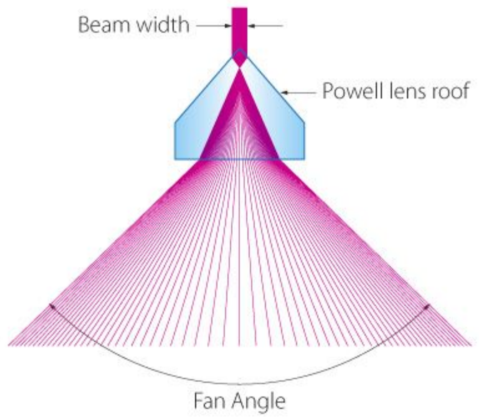
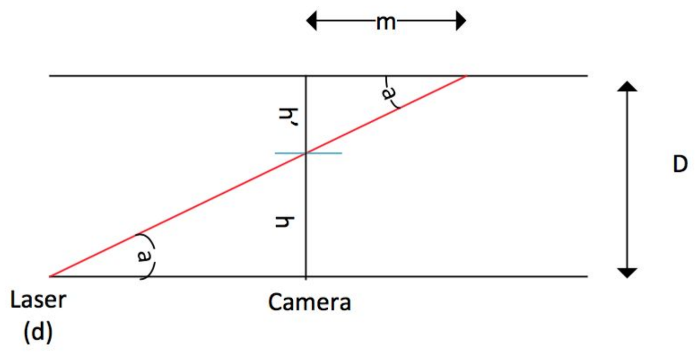


Figure 3: Powell Lens Diagram

On a high level, the scanner maps each of the highlighted points to points in a virtual point cloud. The object is placed against a solid background and a laser is swept across it. The scanner essentially takes photos continuously while scanning, analyzes the position of the laser in each photo and is able to construct a point cloud of the object from these photos. For a given image, the algorithm iterates through each row of the image and finds the brightest point, or the laser, by comparing the new image with the laser to the original image of the object without the laser. These points are added to a vector which is then iterated through, the actual x, y, and z coordinates are computed and the points are added to the point cloud. First, the x coordinate is computed using similar triangles.



D = known distance to reference wall
m = measured distance between reference line and object point

Tan(a) = D/(d+m)
Tan(a) = h'/m

...therefore...

D/(d+m) = h'/m

...therefore...

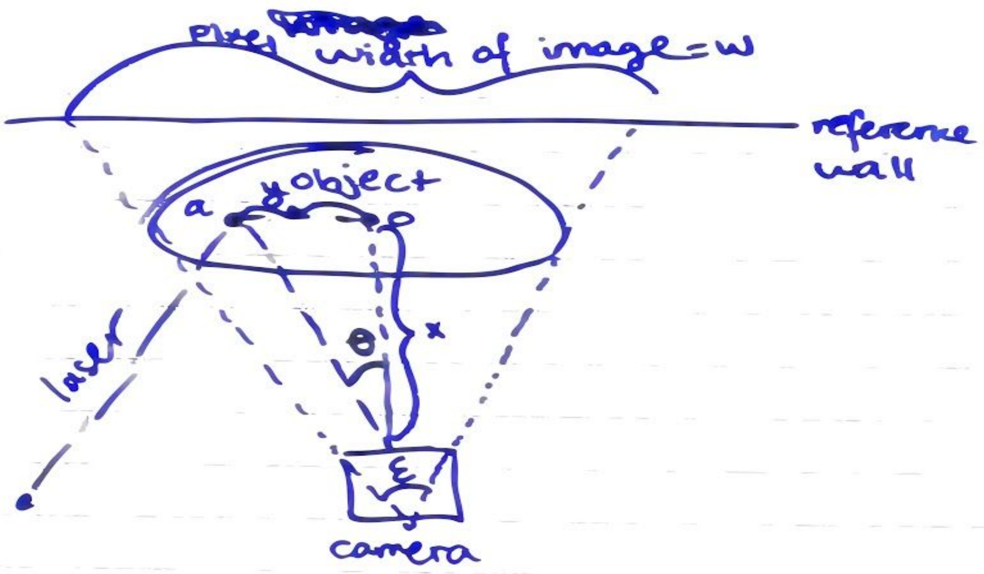
h'=D*m/(d+m)

h = D - h' = D – D*m/(d+m)

"h" is the distance from the camera to the object laser point.

Figure 4:Scanner Point Cloud Collection Equations [12]

The values for D and d are inputted into the program as they remain constant and m is calculated by dividing the pixel distance between the laser position and the center of the camera (half the width of the image) by the number of pixels per centimeter. The value for "h" is the x coordinate of the object highlighted by this portion of the laser. The y and z coordinates are found in the following ways.



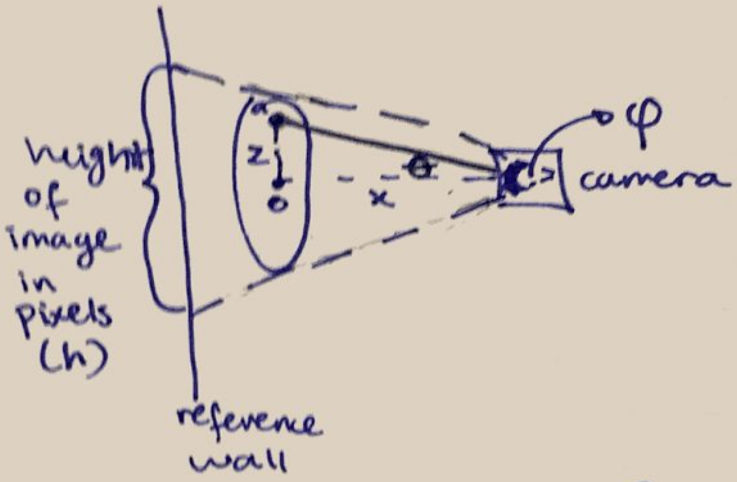
$y = \tan(\theta)x$ } provided y & x are the actual distances (cm). Not the pixel lengths

the pixel length between point "a" and point "o" is v.

$$\frac{\varepsilon}{w} = \frac{\text{degrees}}{\text{pixel}}$$

$$\theta = v \cdot \frac{\varepsilon}{w}$$

therefore $y = \tan(v \cdot \frac{\varepsilon}{w})x$, x is already calculated v can be computed using the image. ε & w are fixed.



$z = x\tan(\theta)$ } provided x & z are the actual distances in cm.

Let v denote the pixel length between point "o" and point "a".

$$\frac{\varphi}{h} = \frac{\text{degrees}}{\text{pixel}}$$

$$\theta = v \cdot \frac{\varphi}{h}$$

$$z = x\tan\left(v \cdot \frac{\varphi}{h}\right)$$

x is calculated earlier, φ & h are constants, and v can be obtained from the image.

These x, y, and z coordinates are computed for each position of the laser in the given vector, and the entire process is repeated for each image taken by the camera, which features the laser at different positions (the laser is manually swept across the object), allowing for a complete point cloud to be obtained.

After the point cloud is completed, the points are placed into “Blender,” an application that allows for extraneous points to be manually removed. The cleaned up point cloud is then placed into Meshlab, where a poisson surface reconstruction is performed.

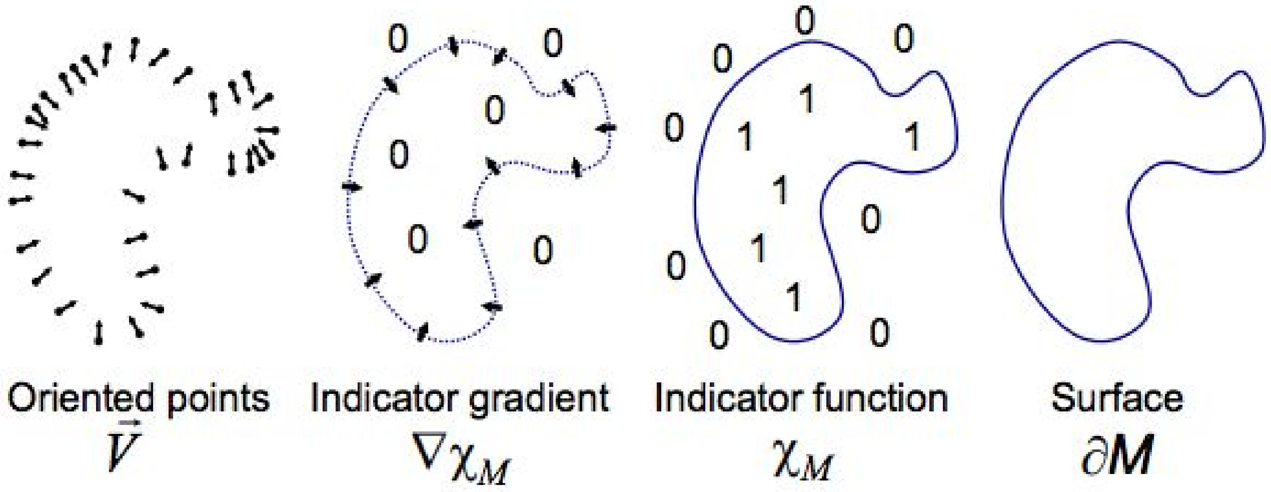

0
1
Oriented points $\vec{V}$
Indicator gradient $\nabla\chi_M$
Indicator function χ_M
Surface ∂M

Figure 5: Illustration of Poisson Reconstruction in 2 dimensions [13]

Essentially, the poisson reconstruction takes the points and “extracts” an isosurface. The steps are illustrated above in Figure 5. From the “Indicator function” step to the “Surface step,” the algorithm, subtracts the portion of the space with zeroes, creating a “line” around the ones. The gradient of this indicator function is essentially zero everywhere (since the function is constant for most of the space), except near the bound surface, where the gradient is the inward normal. So in order to go from the oriented points, to the indicator gradient, one needs to find a function χ_M whose gradient is closest to the vector field. Essentially, one can see that the divergence of the gradient of χ_M is equal to the divergence of the vector field, since divergence is the amount of flux from an infinitesimal volume around a point, the same amount of flux should go through the surface of both the vector function and the gradient of χ_M. This essentially devolves into a Poisson equation, $\nabla^2(f) = g$. So the algorithm backtracks to the set of oriented points. First, given the object M, the object is smoothed using a smoothing function F(q). Without this function, the gradient of the indicator values at the boundary would essentially be infinite, since a pointcloud is piecewise. Let ∂M be the boundary of the solid M, let χM denote the indicator function of M, $N_{\partial M}(p)$ be the inward surface normal at $p \in \partial M$, $F\tilde{}(q)$ be a smoothing filter, and $F\tilde{}_p(q) = F\tilde{}(q-p)$ its translation to the point p. Essentially we obtain the following sets of equations:

$$\begin{aligned}\left.\frac{\partial}{\partial x}\right|_{q_0}(\chi_M * \tilde{F}) &= \left.\frac{\partial}{\partial x}\right|_{q=q_0}\int_M \tilde{F}(q-p)dp \\ &= \int_M \left(-\frac{\partial}{\partial x}\tilde{F}(q_0-p)\right)dp \\ &= -\int_M \nabla\cdot\left(\tilde{F}(q_0-p),0,0\right)dp \\ &= \int_{\partial M}\left\langle\left(\tilde{F}_p(q_0),0,0\right),\vec{N}_{\partial M}(p)\right\rangle dp.\end{aligned}$$

The first equation is constructed from the fact that χM is equal to 1 inside of M and 0 outside of M. The second and equation is obtained by using the property that (∂/∂q)F˜(q− p) = −(∂/∂ p)F˜(q− p). The last equation is obtained using the divergence theorem. Essentially, this proves that: ∇ χM ∗F˜(q0) = ∂M $F˜_p$(q0) $N_{\partial M}$(p)dp. The original pointcloud, S, can now be used to approximate the surface or the vector field V. If ∂M is sectioned into portions Ps ⊂ ∂M, the integral can be approximated by taking the value at a point sample (Ps) and scaling that value by the area of the portion (N is the set of normals calculated from the pointcloud): ∇(χM ∗F˜)(q) = ∑ s∈S Ps $F˜_p$(q) N∂M(p) dp ≈ ∑ s∈S |Ps| $F˜_{ps}$(q) N ≡ V(q). This allows us to obtain our vector field from the original pointcloud. Then we can solve for χ to get the isosurface, using the fact that Δχ˜ = ∇· V [13]. This allows the algorithm to complete the reconstruction, converting the pointcloud into an STL, a printable file.

4.2 Theory of 3D Printer

There are many different models of 3D printers available. For this project, a RepRap Do-It-Yourself kit is being used, that implements a fused filament fabrication (FFF) technology. FFF is a relatively new method of rapid prototyping, and it works by laying down consecutive layers of high-temperature materials, allowing layers to cool and bond before the next layer is applied [14]. Figure 6 below shows the basic premise of an FFF 3D printer. The extruder has a nozzle that retracts the heated filament, which is laid down on the object being printed. The filament is heated using a heater block.

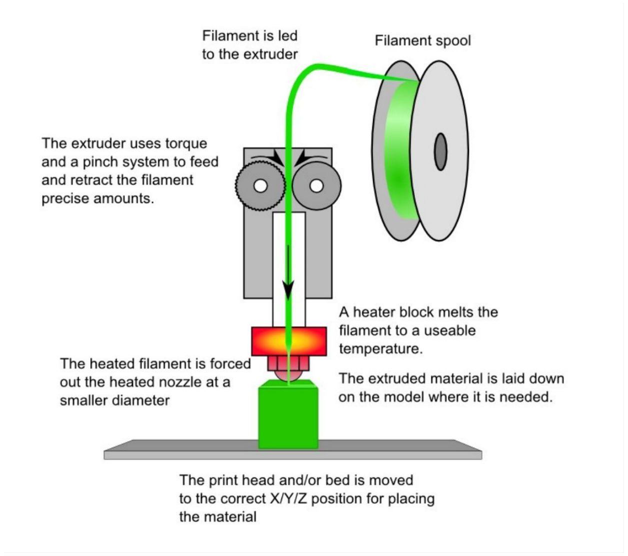

Filament is led to the extruder
Filament spool
The extruder uses torque and a pinch system to feed and retract the filament precise amounts.
A heater block melts the filament to a useable temperature.
The heated filament is forced out the heated nozzle at a smaller diameter
The extruded material is laid down on the model where it is needed.
The print head and/or bed is moved to the correct X/Y/Z position for placing the material

Figure 6: How a Fused Filament Fabrication Model 3D Printer Works

3D printers operate on a 3D coordinate system, commonly known in mathematics as a "Cartesian coordinate system." The printer uses X, Y, and Z coordinates to map to a specific point in the 3D coordinate system. The RepRap Guru Prusa I3 V2 uses a Cartesian XZ head, which means the "bed," or platform upon which the object is printed, moves in the Y direction and the extruder head, which ejects the filament, moves in the X and Z directions [15]. The process for the FFF 3D printer can be seen in Figure 7 below.

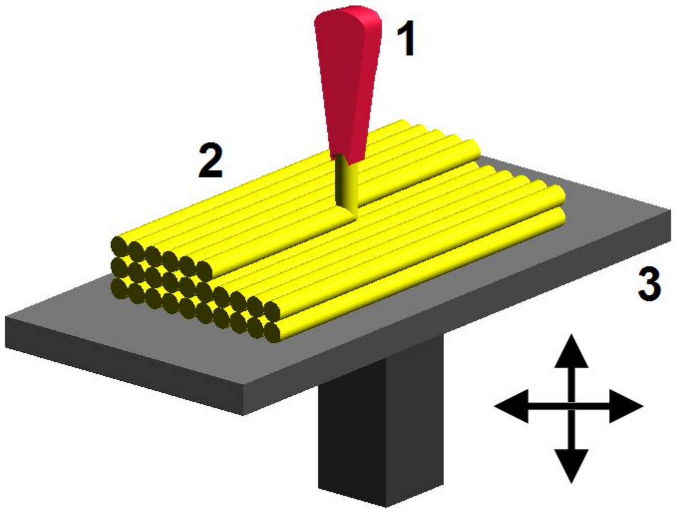


Figure 7: Process of 3D Printing; 1 - 3D Printer Extruder deposits materials; 2- deposited material (modeled part), 3 - controlled movable table

3D printers are generally constructed using fixed rods, timing belts, and pulleys to maneuver the extruder head. Both the timing belts and pulleys are attached to stepper motors, motors that permit extremely precise movements [16].

During the 3D printing process, thermoplastic filament is introduced into the liquefier by mechanical pressure from rollers. The filament is then melted and extruded. Rollers are the only drive mechanism used to deliver the material. Filament is under tensile stress when traveling towards the roller, and under compression when traveling downstream towards the liquefier. Therefore, the driving force in the extrusion process is compressive stress. Figure 8 below shows this.

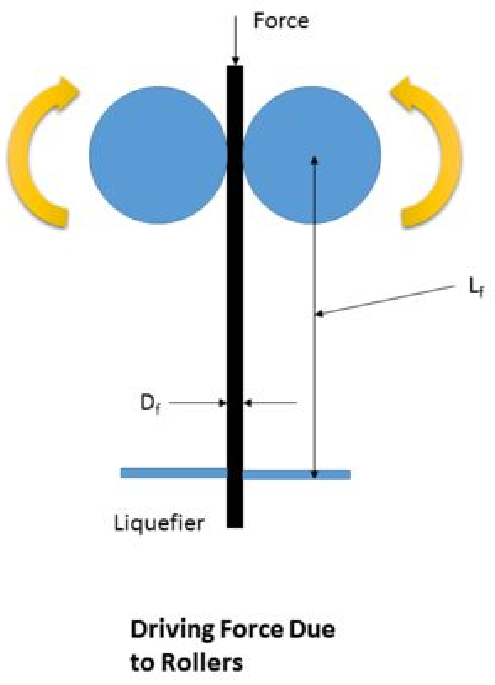


Figure 8: Extruder Driving Force; D_f is the diameter of the filament, L_f is the length of the filament.

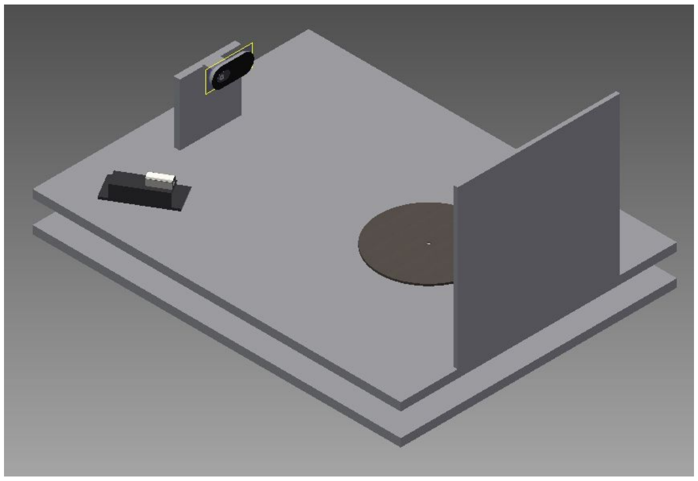
The force needed to extrude the melted filament must be strong enough to overcome the pressure drop across the system, dependent upon the the viscous properties of the melted material and the flow geometry of the liquefier and nozzle. Shear thinning behavior is common to many materials used in FFF 3D printing. Shear thinning is the behavior of fluids whose viscosity decreases under shear strain, modeled by the power law for generalized Newtonian fluids.
Heat input from electrical coil heaters are used to regulate the temperature. In a negative feedback loop, the system continuously adjusts the power supplied to the coils based on the difference in temperature between the desired value and the value detected by the thermocouple [17].

While this printer uses a similar method of laying down consecutive layers of the plastic filament as common 3D printers, including the Makerbot, it is differentiated in the fact that the y platform and x head move independently at each layer, while other printers move the x and y simultaneously. Some benefits of this method of printing includes the ability to customize infills, saving on material costs and enhancing design flexibility. In addition, FFF printing provides for high production speeds. Some drawbacks of this method of 3D printing include accuracy limits as a result of motor accuracy and user calibration along with overhang features.

5.0 Design

5.1 CAD

3D Scanner Design:
The 3D scanner is constructed out of a 24x18 inch plank base, upon which a stepper motor is mounted. Another plank is placed above that through which the stepper motor extends. A 6.5 inch diameter turntable is mounted on the stepper motor shaft. A black backboard is placed behind the stepper motor, which prevents the point cloud from being influenced by external, miscellaneous objects. A camera is placed approximately 12.75 inches from the center of the turntable so that the entirety of the object can be viewed. In addition, the angled line laser is placed approximately 11.95 inches from the center of the turntable. Figure 9 below shows the full design of the 3D scanner. The full drawing with dimensions of the scanner can be seen in Appendix B



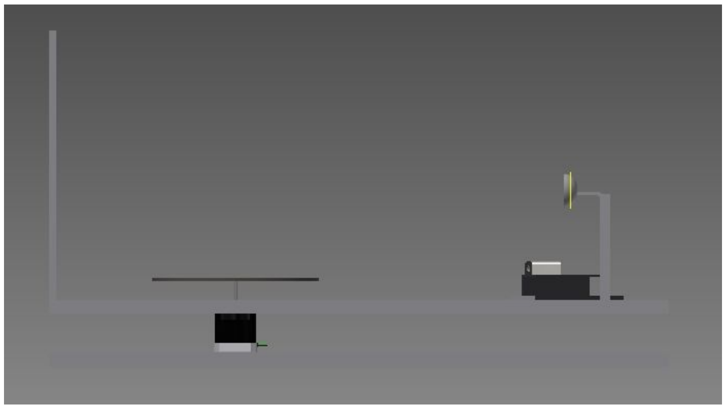
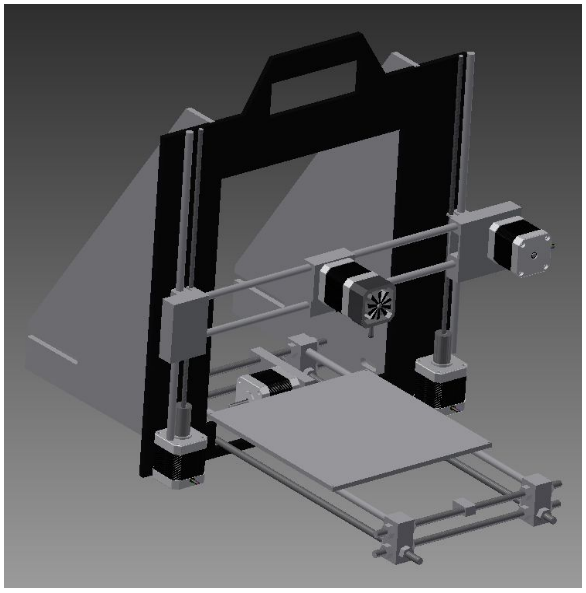


Figure 9: Top and Side Views of the 3D Scanner

3D Printer Design:
The 3D printer is constructed from a RepRap Prusa i3 kit. There are 5 stepper motors on the printer: two controlling the z-axis, one controlling the y-axis, one controlling the x-axis, and one controlling the extruder. The printer uses an XZ cartesian head, while the base moves in the y direction. The z head moves within a single sheet of paper distance from the y platform. The x head moves as far as the edge of the y platform. The printer works by laying down each layer, infilling, and having the z move up one level. This process repeats until the object has been completely 3D printed. Figure 10 below shows the full design of the 3D printer. The full drawing with dimensions of the printer can be seen in Appendix C.



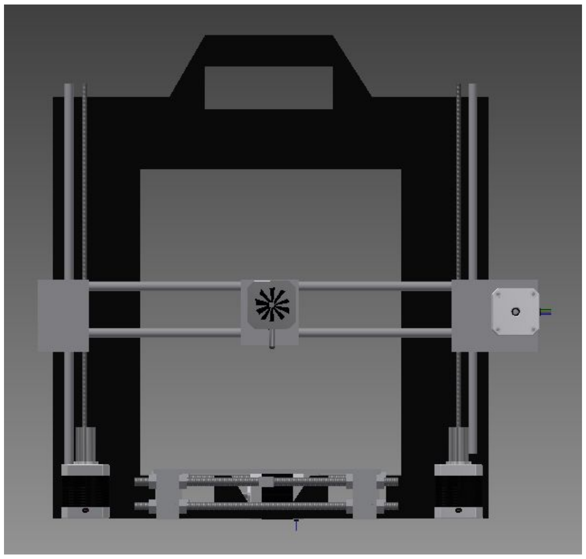
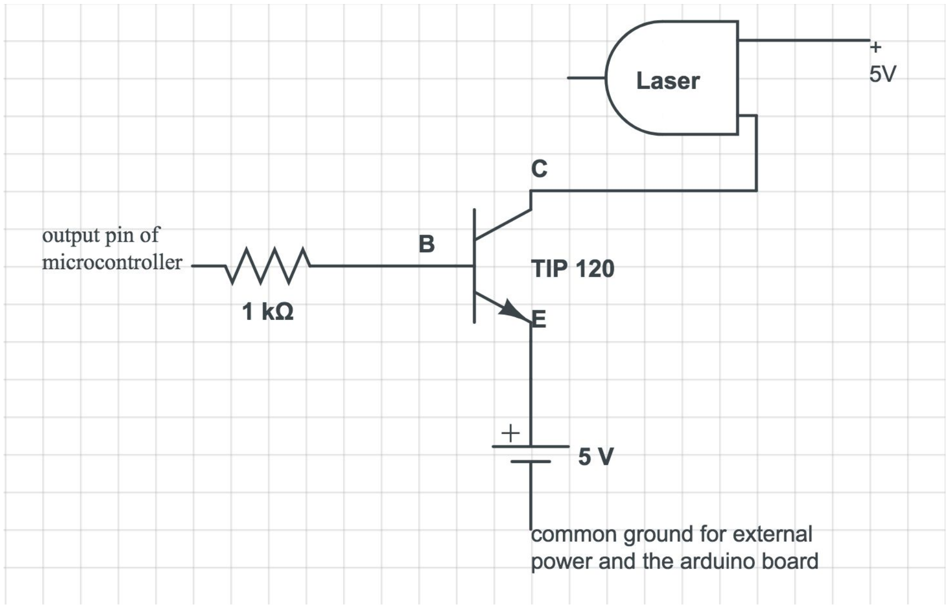


Figure 10: Front and Side View of the 3D Printer

5.2 Circuit Diagrams

Laser Mechanics:

A simple circuit was constructed to power the line laser on and off. The laser is controlled using an Arduino Uno and NPN Tip120 transistor. When the microcontroller outputs a certain voltage through an output pin, the laser turns on and off. The full circuit for the laser can be seen below in Figure 11.



Laser
+
5V
C
output pin of microcontroller
B
TIP 120
1 kΩ
E
+
5 V
common ground for external power and the arduino board

Figure 11: Line Laser Circuit

Stepper Motor Mechanics:

The stepper motor controls the turntable for the 3D scanner.

The stepper motor being used has a standard step of 1.8º, resulting in a 200 step per revolution with 0.043 Nm of holding torque. Holding torque is the "strength" of the stepper motor.

The equation for torque is as follows:

$$\boldsymbol{\tau = I\alpha}$$

τ: torque

I: moment of inertia

α: angular acceleration

The equation for moment of inertia is as follows:

$$\mathbf{I = \tfrac{1}{2} (MR^2)}$$

I: moment of inertia

M: mass

R: radius

The equation for angular acceleration is as follows:

$$\boldsymbol{\alpha = a/R}$$

α: angular acceleration

a: acceleration

R: radius

From these equations we develop the following equation:

$$\boldsymbol{\tau = \tfrac{1}{2} (MR^2) * a/R}$$

We can use this equation to calculate the maximum force that the stepper motor can sustain. If we have an object with radius = 10 cm,

$$0.043 \text{ Nm} = \tfrac{1}{2} (\text{mass})(0.1\text{m})^2 * \text{acceleration}/0.1\text{m}$$
$$0.86 \text{ N} = \text{mass} * \text{acceleration}$$

Because Force is equal to mass * acceleration, this shows that the stepper motor can support the weight of the turntable and the objects placed upon it for scanning.

The stepper motor is comprised of two coils, a magnetic north pole, and a magnetic south pole. This allows the stepper motor to rotate. A stepper motor EasyDriver is being used to control the stepper motor, and it is powered using an Arduino Uno microcontroller. The EasyDriver sends alternating signals to the stepper motor based on the direction of current flow.

The full electrical design for the turntable stepper motor can be seen below in Figure 12.

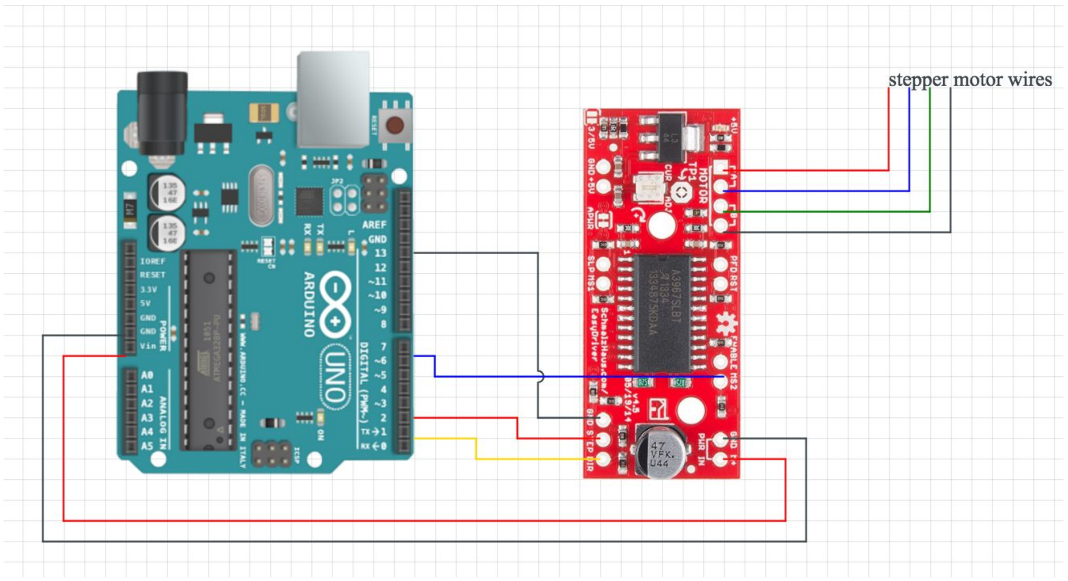


Figure 12: Stepper Motor Circuit

5.3 Scanner Code

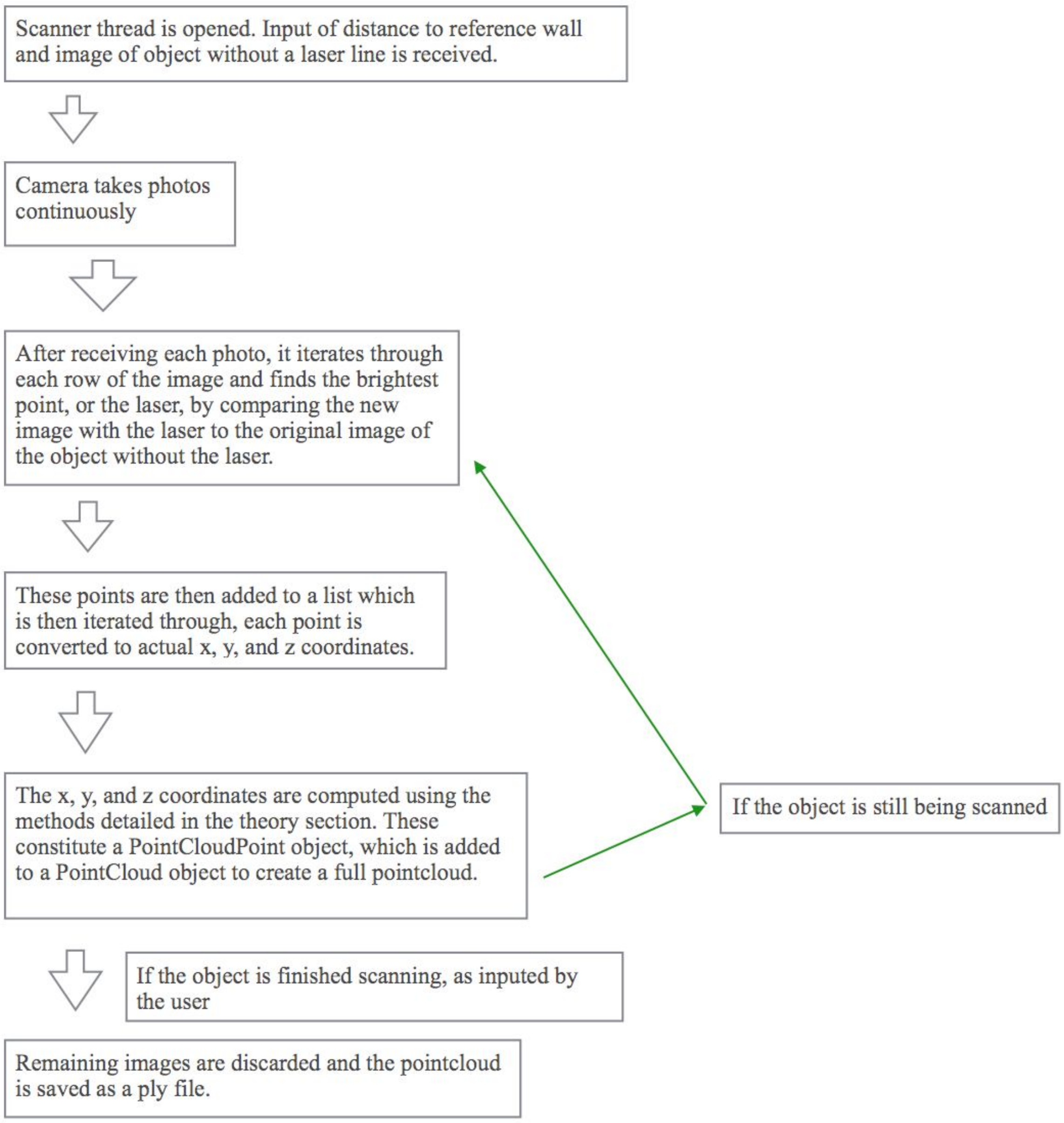

Scanner thread is opened. Input of distance to reference wall and image of object without a laser line is received.
Camera takes photos continuously
After receiving each photo, it iterates through each row of the image and finds the brightest point, or the laser, by comparing the new image with the laser to the original image of the object without the laser.
These points are then added to a list which is then iterated through, each point is converted to actual x, y, and z coordinates.
The x, y, and z coordinates are computed using the methods detailed in the theory section. These constitute a PointCloudPoint object, which is added to a PointCloud object to create a full pointcloud.
If the object is still being scanned
If the object is finished scanning, as inputed by the user
Remaining images are discarded and the pointcloud is saved as a ply file.

Figure 13: Flowchart of Code

6.0 Results

The performances and features for the 3D scanner and 3D printer can be summarized as follows.

6.1 3D Scanner Results

The point cloud acquisition rate of the 3D scanner is dependent on the speed of acquisition of the webcam. This scanner uses a Logitech C270 Desktop Webcam and captures up to 1280 x 720 pixels of data with each scan. The total time the scanner takes to collect and save a point cloud to the PC varies between 45 seconds to 60 seconds.

Scans of a plastic ring were taken at different distances from the camera to the ring that was mounted on the wall to test the intensity at which the scan quality was the best. The intensity equation is $I = P/4\pi r^2$, where I is intensity, P is power, and r is the distance from the object. The power of the laser in this test was held constant at 5mW. The intensity results for the distances test can be seen below.

Table 1 - Intensity of Laser For Plastic Ring Tests

Test Number	Power (W)	Distance (m^2)	Intensity (W/m^2)
1	0.0050	1.5240	0.0002
2	0.0050	0.9398	0.0005
3	0.0050	0.8255	0.0006
4	0.0050	0.6731	0.0009
5	0.0050	0.4953	0.0016
6	0.0050	0.3366	0.0035

The results for the 6 intensity tests are as follows.

For Test 1, the intensity of 0.0002 W/m^2 for the laser on the ring was too low for the scanner to collect any data.

For Test 2, the intensity of 0.0005 W/m^2 for the laser rendered the following images. While the ring shape was seen in the scan, the circular outline and center hole were not captured well in the point cloud, creating a low quality 3D representation of the ring.

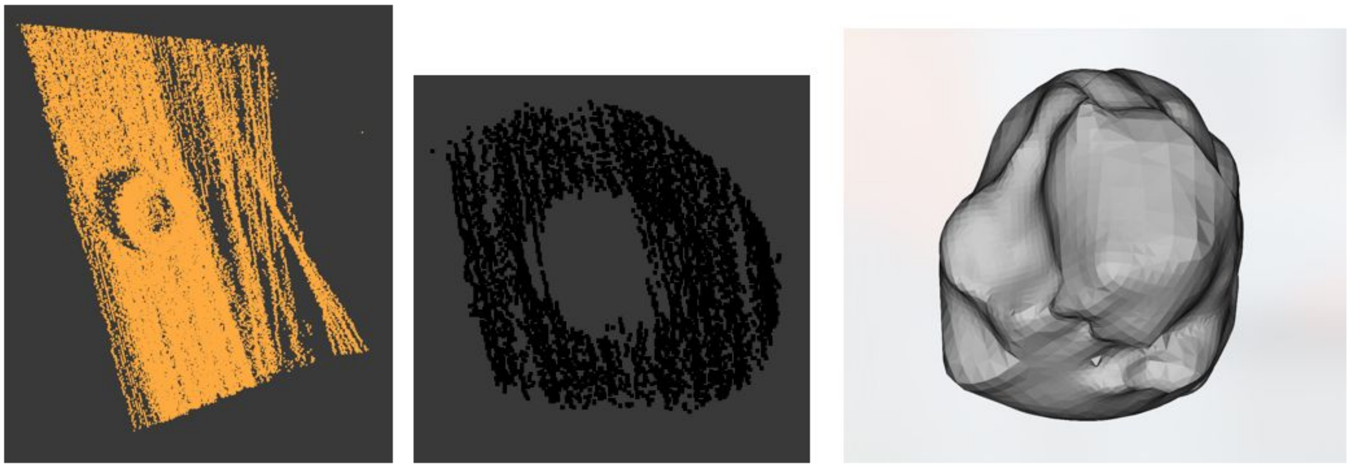


Figure 14: Test 2 of Plastic Ring Scan Test; Intensity = 0.0005 W/m^2

For Test 3, the intensity of 0.0006 W/m^2 for the laser rendered the following images. Again while the ring shape was seen in the scan, the point cloud was distorted. There was no hole capturable in the 3D modeling, and the circular outline was not too noticeable.

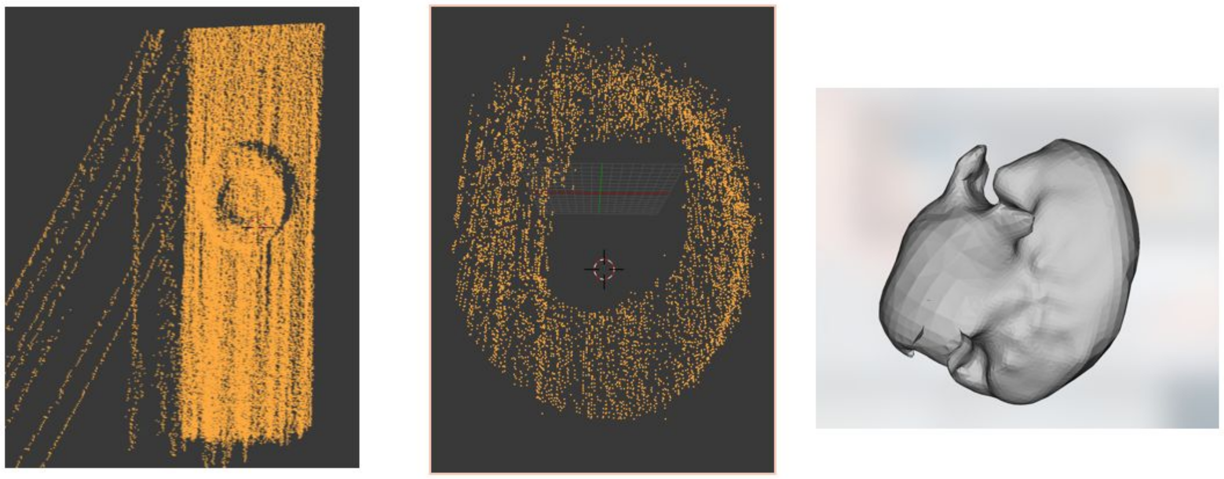


Figure 15: Test 3 of Plastic Ring Scan Test; Intensity = 0.0006 W/m^2

For Test 4, the intensity of 0.0009 W/m^2 for the laser rendered the following images. The captured ring point cloud looked more clean than the previous tests. However, no ring hole was rendered in the 3D modeling, but the model did create a strong circular outline.

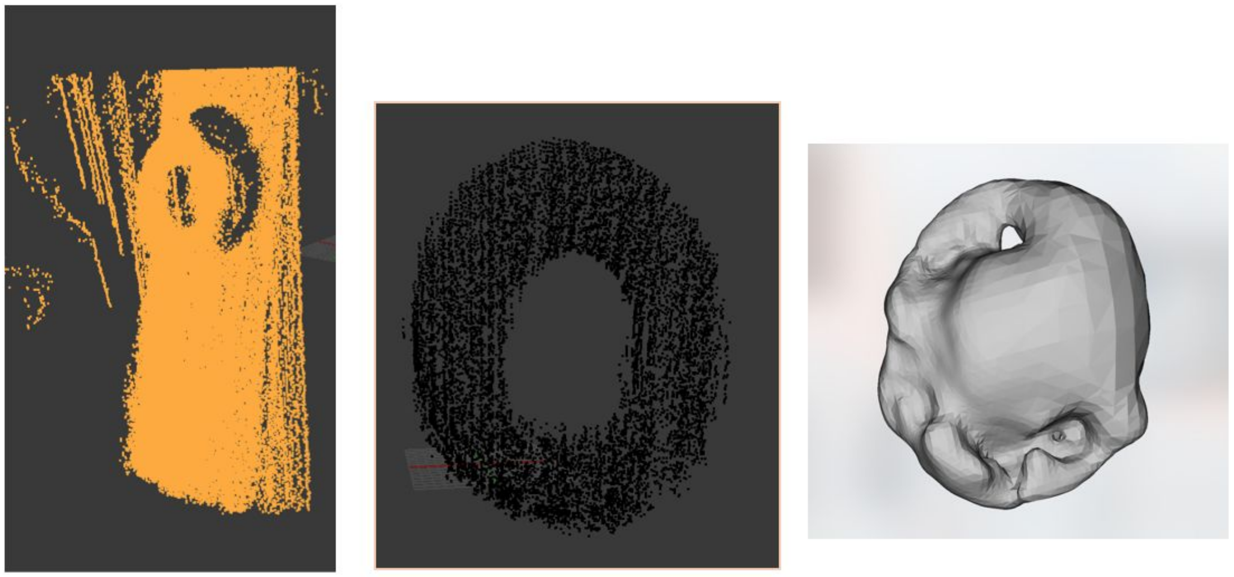


Figure 16: Test 4 of Plastic Ring Scan Test; Intensity = 0.0009 W/m^2

For Test 5, the intensity of 0.0016 W/m^2 for the laser rendered the following images. The hole was rendered in the 3D modeling and circular outline was present. However, the model was distorted and pulled, making the model too wide for printing the ring.

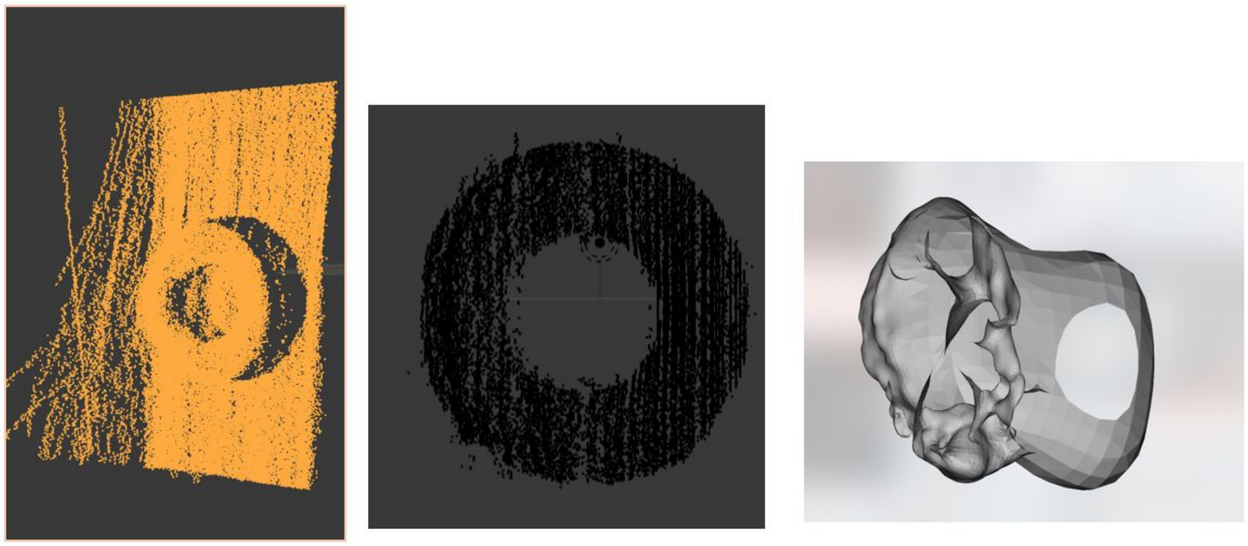


Figure 17: Test 5 of Plastic Ring Scan Test; Intensity = 0.0016 W/m^2

For Test 6, the intensity of 0.0035 W/m^2 for the laser rendered the following images. The hole was rendered in the 3D modeling and circular outline was present. The 3D model created was the best representation of the ring thus far. This model was able to be printed as shown below.

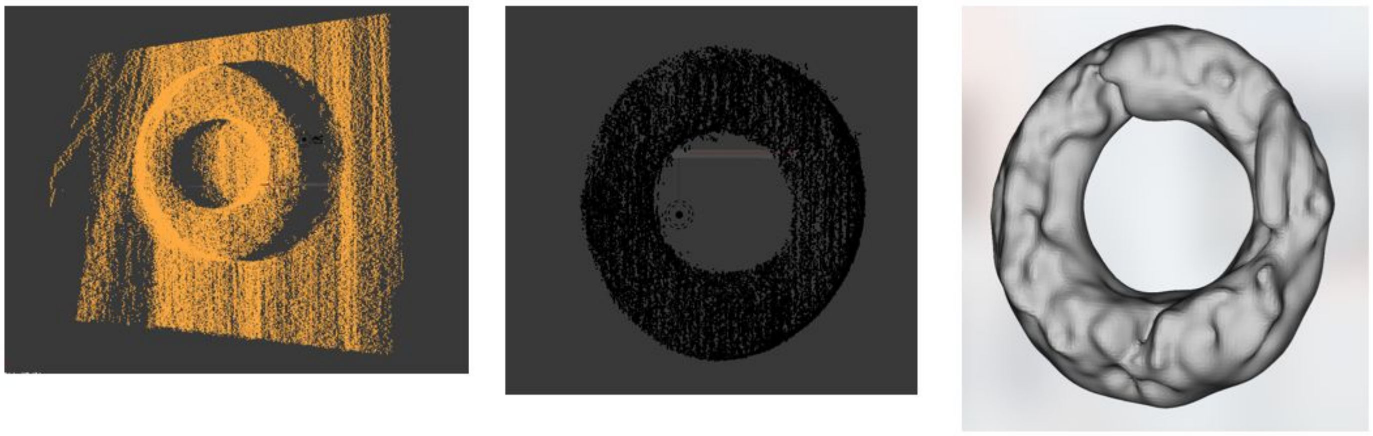


Figure 18: Test 6 of Plastic Ring Scan Test; Intensity = 0.0035 W/m^2

These results show that the smaller the distance between the object and the camera and the higher the intensity of the laser on the object, the better the quality of the 3D model. The 3D model that was created during Test 6 was able to be printed using the Prusa i3 printer that was built. These are the resulting images:

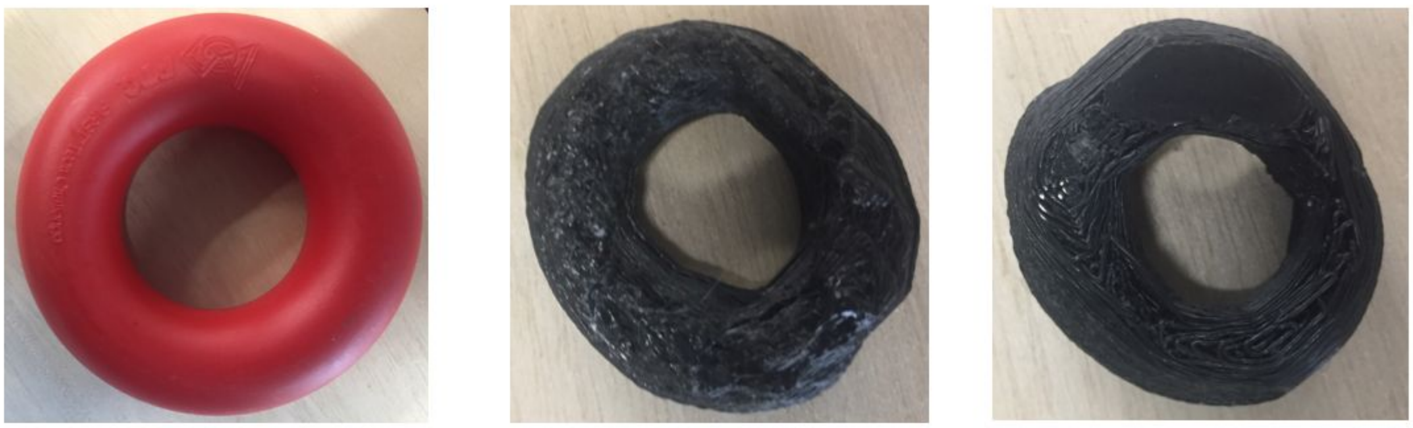


Figure 19: Plastic Ring Print Results at Scan of Intensity = 0.0006 W/m^2; Red Ring is the original

The quality of the print shows that scanning and printing objects is possible. However, the print also shows that there is much area for improvement. The accuracy of the scanner depends on a number of factors: the optical quality of the webcam being used to capture the point cloud (the resolution and lens radial distortion), the angle of the laser in respect to the angle of the camera lens, as discussed in Section 4.1, and finally the accuracy depends on the reflective properties of the scanned surface. Acquiring a highly specular surface is in fact rather difficult.

There are a limitations to the scanner at this stage of the project. Currently, the scanner can only attain scans of static objects from one view. However, in time this can be changed by implementation of the turntable. In order to create a 3D model of the entire object from all angles, the turntable must turn a certain set of degrees, the scan must be taken, the turntable must be turned again, another scan must be taken, and this process repeats. Then, in Meshlab, the stitching tool can be used to combine these different scans to create one complete scan of the object from all angles.

6.2 3D Printer Results

The Prusa i3 mk2 printer was able to attain a 0.012 mm accuracy on the XZ axis and 0.004 mm accuracy on the Y axis. The printer was tested at multiple different temperatures (both the extruder temperature as well as the bed temperature) in order to determine the ideal print temperature for both. These temperatures were found to be 210 ºC for the extruder temperature and 80 ºC for the bed temperature. The printer has a print size of 200 x 200 x 180 mm. The printer uses 5 NEMA 17 stepper motors.

The printer was tested with two types of infill methods. The rectangular infill was used at first as shown in the SF Giants Logo and cube that were printed.

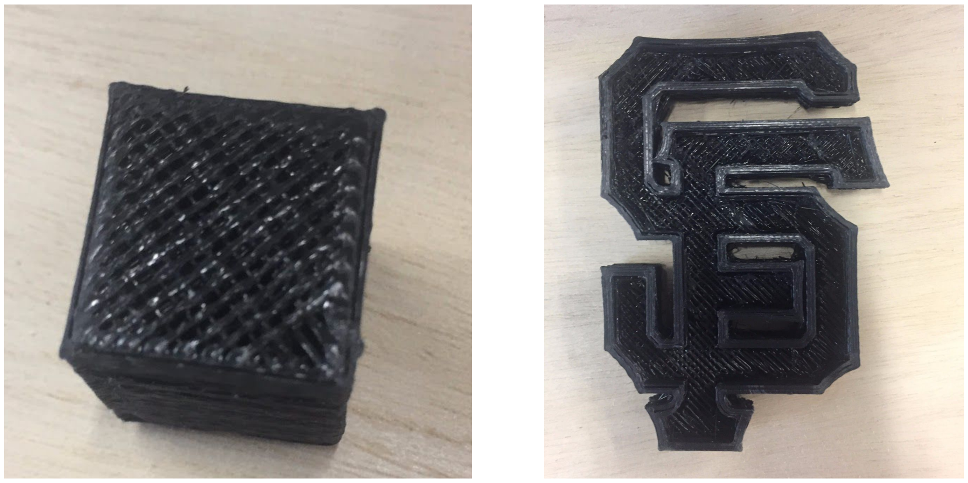


Figure 20: Print Results Using Rectangular Infill Method

While this method of infill worked consistently, the honeycomb infill was attempted at well. This infill method proved to have the fastest print times, save the most material, and be stronger than the rectangular method.

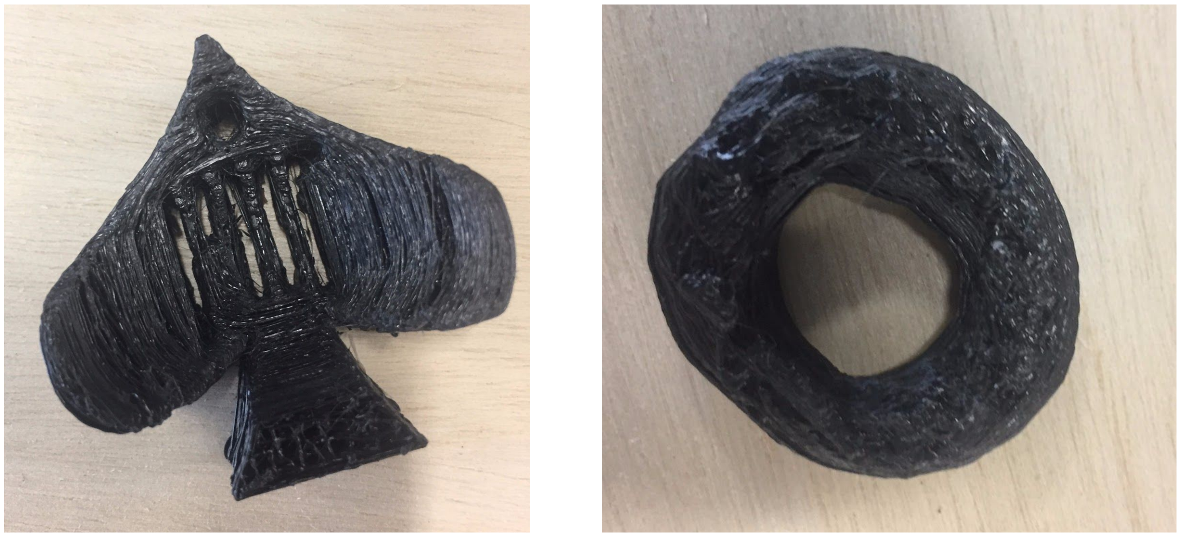


Figure 21: Print Results Using Honeycomb Infill Method

There was a challenge with the printer. This main challenge was the extruder inconsistency. At times the filament would get caught in the extruder and fail to extrude during in the print, resulting in gaps in the object while it was being printed. In the future, this problem should be fixed by opening up the extruder to see if there are any issues with the driver. However, as a whole, the printer managed to work well and produce some high quality prints for the project.

7.0 Conclusion

Overall, this project resulted in a functioning 3D scanner able to process images and reconstruct the physical objects into virtual models that could be printed by a functioning 3D Cartesian printer. The printer is able to construct objects up to 20.32cm × 20.32cm × 17.78cm using PLA, ABS, Nylon, Flexible PLA, PVA, HIPS, Wood, PET filaments. The scanner's accuracy varies with distance to the object and can transform point clouds into an STL file using a poisson reconstruction.

The fundamental issue with the scanner is the ability to only capture one face of an object. In order to reconstruct a complete, complex object the scanner needs to take at least two scans which then need to be manually stitched together. Furthermore, the scanner requires a manual measurement from the camera to the reference wall, which is then rounded to the nearest centimeter in the program. This creates an error of +/- 1cm, which warps the x, y, and z coordinates, since the y and z coordinates are multiplied by the x coordinate which is calculated by multiplying the measured reference distance by constants. Furthermore, the calculations assume that the camera and the laser are in the same plane of reference, parallel to the reference wall. This further warps the point clouds when using different types of cameras, for example when upgrading the camera resolution.

The impressive part about the scanner is that it essentially proves that computer vision is possible. The computer can "see" the objects virtually and create a depth map of its surroundings. This potentially could allow for computers to interact with the physical environment, paving the way for smart robots, augmented reality, and more.

8.0 Next Steps

With regards to the printer, we would make modifications to the extruder so that it is able to "grab" the filament better. The printer currently has difficulties printing continuously as the filament gets stuck in a certain position at times. A stepper motor "prong" that fits more tightly around the filament would eliminate this problem and allow for continuous printing. As to the scanner, we would continue to make modifications to obtain "better" point clouds. We would implement the turntable system that we initially set out to make to allow for a full reconstruction without stitching point clouds together. It also eliminates the need for the laser to sweep across the object, allowing for an easier reconstruction of the object. Furthermore, we would use a thinner laser and a higher resolution camera. Upgrading the camera and the laser would allow better detection of the isolated pixel, creating more detailed and accurate point clouds.

Acknowledgments
Thank you to Dr. Dann for guiding and supporting us through our many ASR endeavors this year and teaching us to never give up when facing challenges.

Thanks to Mr. Ward for his helpful advice, explanations of how to use the tools in the toolshop, unlocking of the darkroom, and most importantly his witty remarks.

Thanks to our ASR classmates for their assistance and constant entertainment.

Thanks to the red ring that saved our project and gave us hope.

9.0 Bibliography

[1] McCarthy, Patrick. "5 DIY 3D Scanners to Watch." *Make:*, January 15, 2015. Accessed February 6, 2017. http://makezine.com/2015/01/15/5-diy-3D-scanners-to-watch/.

[2] Konecny, Christina. "A Brief History of 3D Scanning." *Matter and Form* (blog). Entry posted December 10, 2014. Accessed February 6, 2017. https://matterandform.net/blog/a-brief-history-of-3D-scanning.

[3] *Types of 3D Scanners and 3D Scanning Technologies*. Tampa, FL, n.d. Accessed February 6, 2017. https://www.ems-usa.com/tech-papers/3D%20Scanning%20Technologies%20.pdf.

[4] "Lidar." In *Wikipedia*. Accessed May 1, 2017. https://en.wikipedia.org/wiki/Lidar.

[5] "Types of 3D Printers or 3D Printing Technologies Overview." 3D Printing from Scratch. Last modified 2015. Accessed January 12, 2017. http://3Dprintingfromscratch.com/common/types-of-3D-printers-or-3D-printing-technologies-overview/.

[6] "Rapid Prototyping." PROTOSYS Technologies. Last modified 2005. Accessed January 12, 2017. http://www.protosystech.com/rapid-prototyping.htm.

[7] "Industrial Design and Manufacturing." Artec3D. Last modified 2016. Accessed February 6, 2017. https://www.artec3D.com/applications.

[8] Apis Cor. Accessed May 3, 2017. http://apis-cor.com/en/.

[9] "3D Printing Applications." 3D Printing Industry. Accessed January 12, 2017. https://3Dprintingindustry.com/3D-printing-basics-free-beginners-guide/applications/.

[10] "How Lasers Work." Lawrence Livermore National Laboratory. Accessed February 6, 2017. https://lasers.llnl.gov/education/how_lasers_work.

[11] "Powell Lenses." Altechna. Accessed March 9, 2017. http://www.altechna.com/product_details.php?id=1239.

[12] Barry, Andy. *Makerscanner*. Github. Accessed March 17, 2017. https://github.com/andybarry/makerscanner.

[13] Kazhdan, Michael, Matthew Bolido, and Hugues Hoppe. *Poisson Surface Reconstruction*. N.p.: Eurographics Symposium on Geometry Processing, 2006.

[14] Feeney, David. "FFF Vs. SLA Vs. SLS: 3D Printing." SD3D. Last modified August 29, 2013. Accessed February 7, 2017. http://www.sd3D.com/fff-vs-sla-vs-sls/.

[15]"12 Best 3D Printers Launched in 2016." 3DPrinting.com. Last modified November 19, 2016. Accessed February 7, 2017. http://3Dprinting.com/3Dprinters/best-3D-printers/#different-types-of-3D-printers.

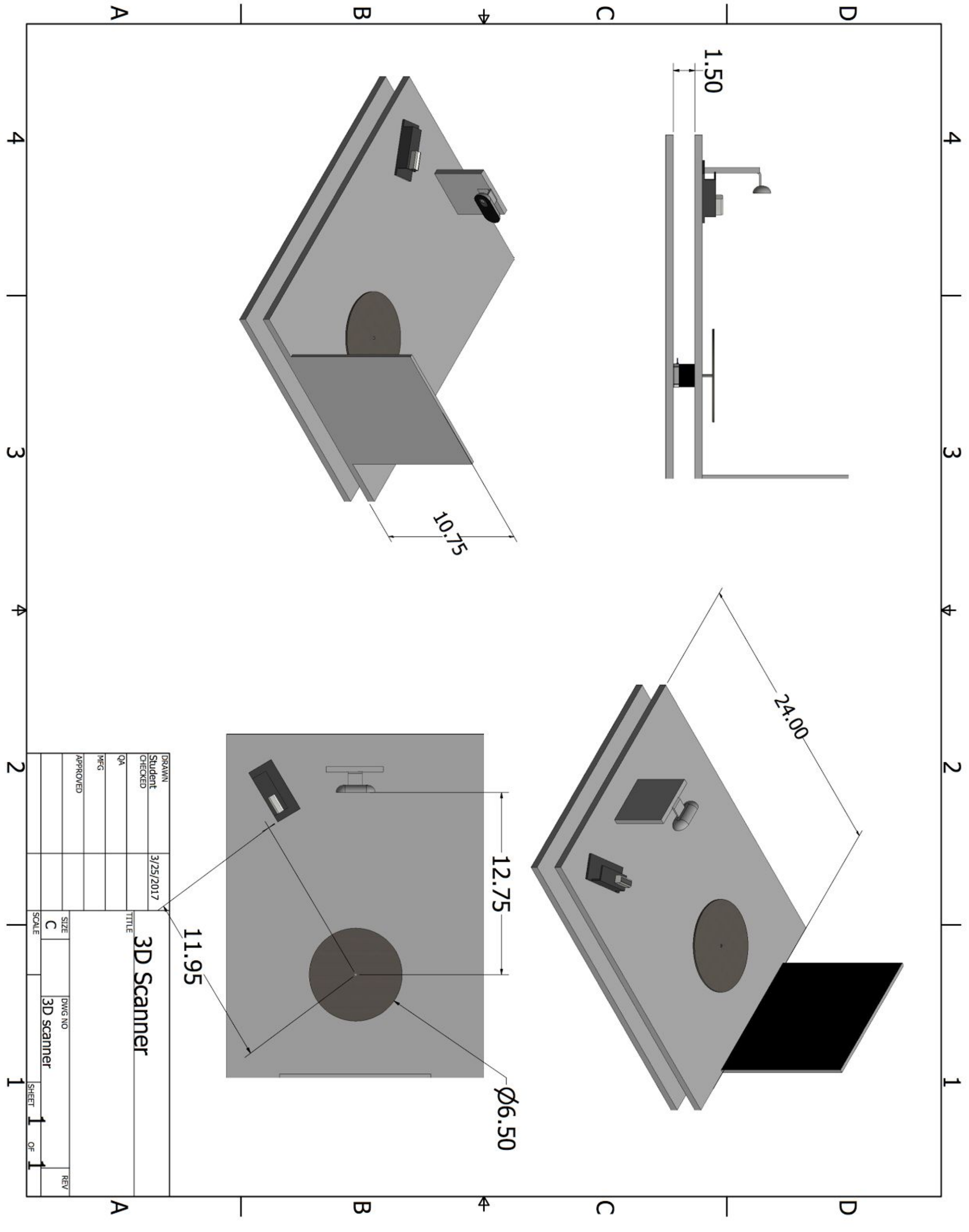
[16] "3D Printers - How Do They Work?" 3DPrintingForBeginners.com. Accessed February 7, 2017. http://3Dprintingforbeginners.com/3D-printing-technology/.

[17] *Wikipedia*. Accessed February 7, 2017. https://en.wikipedia.org/wiki/Fused_filament_fabrication.

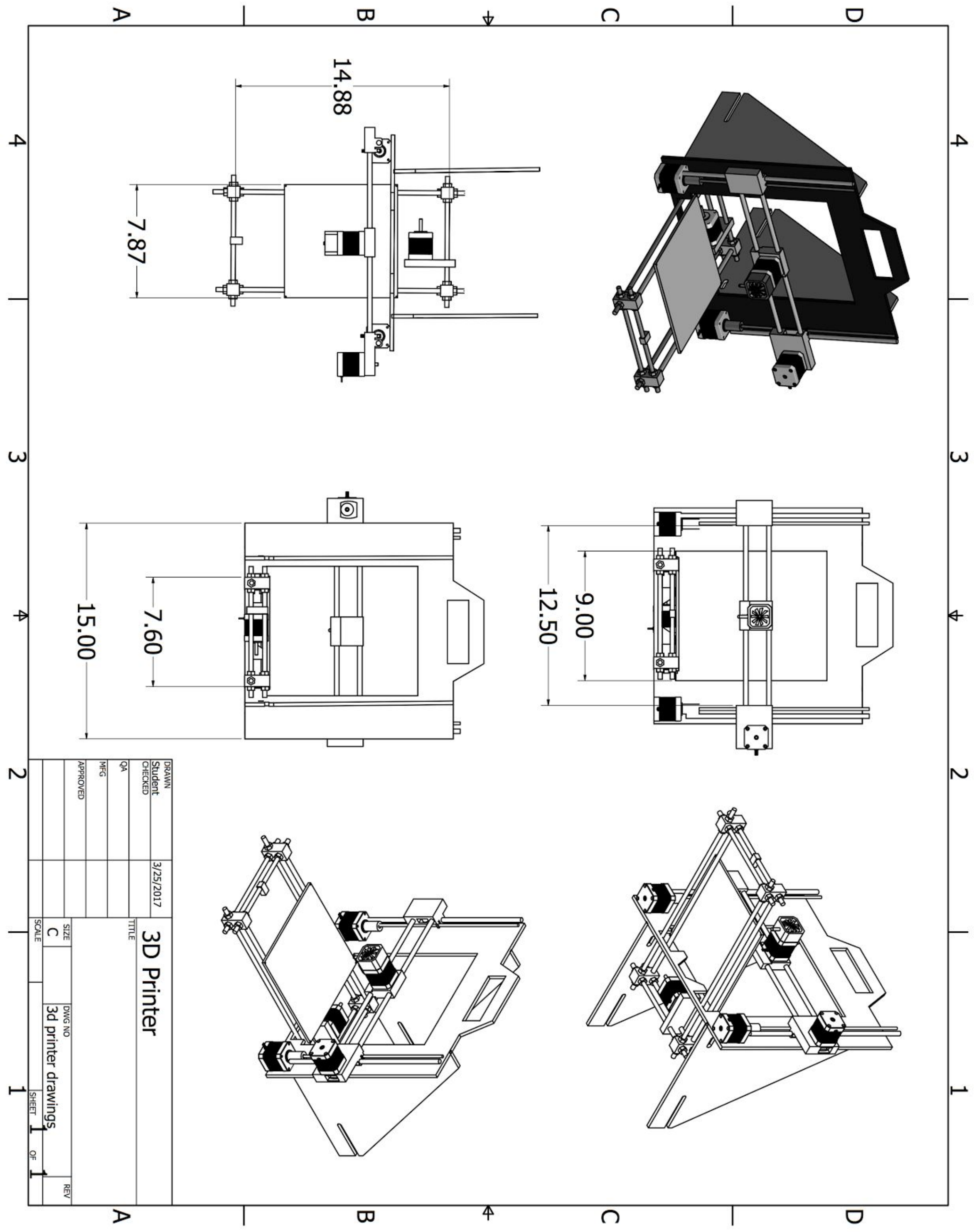
10.0 Appendix A: *Parts List*

Part Description	**What It's Needed For**	**Cost**	**Where You'll Buy It**
Stepper Motor	The object turns on the turntable. It turns a certain amount of steps before the camera takes an image. The stepper motor will be used to turn the turntable for this.	Free	Available in the Lab
Arduino UNO	The Arduino UNO is used to control the stepper motor and the lasers.	Free	Available in the Lab
NPN Transistors	The transistors are used to turn the lasers on and off depending on how much the stepper motor turntable has turned.	Free	Available in the Lab
1KΩ Resistors	The resistors are used in the circuit for the lasers.	Free	Available in the Lab
Motor Driver	The motor driver controls the movement of the stepper motor.	Free	Available in the Lab
Red Laser Modules	The red lasers modules are used to construct the point cloud after the 2D images have been collected. Based on the light intensity of the red laser in the images, we can create the model and detect the curvature of the object.	$25/each	Apinex.com
Logitech Webcam	The Logitech Webcam is used to take the 2D images of the object while it turns on the stepper motor. These 2D images will be converted into a 3D model using the point cloud library	$19.99	Amazon.com
RepRap 3D Printer	The RepRap 3D will be used to test the 3D scanner and print the objects that have been scanned.	$328.99	3DPrinting.com

8.2 Appendix B: *3D Scanner CAD*



8.3 Appendix C: *3D Printer CAD*



8.4 Appendix D: *Scanner Code*

```
/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
* ActiveStereo.cpp
*/

#include "ActiveStereoApp.h"
//(*AppHeaders
#include "ActiveStereoMain.h"
#include <wx/image.h>
//*)
IMPLEMENT_APP(ActiveStereoApp);

bool ActiveStereoApp::OnInit()
{
	//(*AppInitialize
	bool wxsOK = true;
   wxInitAllImageHandlers();
	if ( wxsOK )
	{
		ActiveStereoFrame* Frame = new ActiveStereoFrame(0);
		Frame->Show();
		SetTopWindow(Frame);
	}
	//*)
	return wxsOK;

}

/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
```

```
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*ActiveStereoMain.cpp
*/

#include "ActiveStereoMain.h"
#include <wx/msgdlg.h>
#include <wx/numdlg.h>

//(*InternalHeaders(ActiveStereoFrame)
#include <wx/string.h>
#include <wx/intl.h>
//*)


//helper functions
enum wxbuildinfoformat {
	short_f, long_f };

wxString wxbuildinfo(wxbuildinfoformat format)
{
	wxString wxbuild(wxVERSION_STRING);

	if (format == long_f )
	{
#if defined(__WXMSW__)
	wxbuild << _T("-Windows");
	#define PIXMAPS_DIR ""
#elif defined(__UNIX__)
	wxbuild << _T("-Linux");
#endif

#if wxUSE_UNICODE
	wxbuild << _T("-Unicode build");
#else
	wxbuild << _T("-ANSI build");
#endif // wxUSE_UNICODE
	}
```

```
        return wxbuild;
}

//(*IdInit(ActiveStereoFrame)
const long ActiveStereoFrame::ID_STATICTEXT3 = wxNewId();
const long ActiveStereoFrame::ID_STATICTEXT4 = wxNewId();
const long ActiveStereoFrame::ID_STATICTEXT1 = wxNewId();
const long ActiveStereoFrame::ID_FPS_LABEL = wxNewId();
const long ActiveStereoFrame::ID_BUTTON4 = wxNewId();
const long ActiveStereoFrame::ID_STATICTEXT7 = wxNewId();
const long ActiveStereoFrame::ID_SLIDER3 = wxNewId();
const long ActiveStereoFrame::ID_STATICTEXT5 = wxNewId();
const long ActiveStereoFrame::ID_SLIDER1 = wxNewId();
const long ActiveStereoFrame::ID_STATICTEXT6 = wxNewId();
const long ActiveStereoFrame::ID_BUTTON3 = wxNewId();
const long ActiveStereoFrame::ID_BUTTON1 = wxNewId();
const long ActiveStereoFrame::ID_STATICLINE1 = wxNewId();
const long ActiveStereoFrame::ID_TEXTCTRL1 = wxNewId();
const long ActiveStereoFrame::ID_PANEL1 = wxNewId();
const long ActiveStereoFrame::idMenuQuit = wxNewId();
const long ActiveStereoFrame::idMenuChangeCamera = wxNewId();
const long ActiveStereoFrame::idMenuAbout = wxNewId();
const long ActiveStereoFrame::ID_STATUSBAR1 = wxNewId();
//*)

BEGIN_EVENT_TABLE(ActiveStereoFrame,wxFrame)
    //(*EventTable(ActiveStereoFrame)
        //*)

    EVT_COMMAND(wxID_ANY, IMAGE_UPDATE_EVENT, ActiveStereoFrame::UpdateImage)
    EVT_COMMAND(wxID_ANY, DISPLAY_TEXT_EVENT, ActiveStereoFrame::DisplayText)
        EVT_COMMAND(wxID_ANY, WRITE_TO_FILE_EVENT, ActiveStereoFrame::WriteToFile)
    EVT_COMMAND(wxID_ANY, SCAN_PROGRESS_EVENT, ActiveStereoFrame::UpdateScanProgress)
    EVT_COMMAND(wxID_ANY, SCAN_FINISHED_EVENT, ActiveStereoFrame::ScanFinished)

        EVT_UPDATE_UI(ID_FPS_LABEL, ActiveStereoFrame::UpdateFps)

END_EVENT_TABLE()

// Much of this is wxSmith generated code that creates (most of) the GUI.
ActiveStereoFrame::ActiveStereoFrame(wxWindow* parent,wxWindowID id)
{
    //(*Initialize(ActiveStereoFrame)
        wxMenuItem* MenuItem2;
        wxMenuItem* MenuItem1;
        wxFlexGridSizer* FlexGridSizer1;
        wxMenu* Menu1;
```

```
	wxBoxSizer* BoxSizer2;
	wxStaticBoxSizer* staticBoxSizerSettings;
	wxBoxSizer* topBoxSizer;
	wxBoxSizer* BoxSizer1;
	wxMenuBar* MenuBar1;
	wxMenu* Menu2;

	Create(parent, wxID_ANY, _("MakerScanner v0.3.2"), wxDefaultPosition, wxDefaultSize,
wxDEFAULT_FRAME_STYLE, _T("wxID_ANY"));
   SetClientSize(wxSize(1009,746));
   SetMinSize(wxSize(1009,746));
	Panel1 = new wxPanel(this, ID_PANEL1, wxPoint(120,232), wxDefaultSize, wxTAB_TRAVERSAL,
_T("ID_PANEL1"));
	BoxSizer1 = new wxBoxSizer(wxVERTICAL);
	topBoxSizer = new wxBoxSizer(wxHORIZONTAL);
	headBoxSizer = new wxBoxSizer(wxVERTICAL);
   staticBoxSizerStatus = new wxStaticBoxSizer(wxHORIZONTAL, Panel1, _("System Status"));
	flexGridStatus = new wxFlexGridSizer(0, 2, 0, 0);
   lblStaticCameraConnected = new wxStaticText(Panel1, ID_STATICTEXT3, _("Camera: "),
wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT3"));
	flexGridStatus->Add(lblStaticCameraConnected, 0,
wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
	lblCameraConnected = new wxStaticText(Panel1, ID_STATICTEXT4, _("Disconnected"),
wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT, _T("ID_STATICTEXT4"));
	flexGridStatus->Add(lblCameraConnected, 1,
wxALL|wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL, 5);
	StaticText1 = new wxStaticText(Panel1, ID_STATICTEXT1, _("FPS: "), wxDefaultPosition,
wxDefaultSize, 0, _T("ID_STATICTEXT1"));
   flexGridStatus->Add(StaticText1, 0, wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
	lblFPS = new wxStaticText(Panel1, ID_FPS_LABEL, _("0.0"), wxDefaultPosition, wxDefaultSize, 0,
_T("ID_FPS_LABEL"));
   flexGridStatus->Add(lblFPS, 1, wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
   staticBoxSizerStatus->Add(flexGridStatus, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	butCameraConnect = new wxButton(Panel1, ID_BUTTON4, _("Camera Connect"),
wxDefaultPosition, wxDefaultSize, 0, wxDefaultValidator, _T("ID_BUTTON4"));
   staticBoxSizerStatus->Add(butCameraConnect, 1,
wxALL|wxALIGN_TOP|wxALIGN_CENTER_HORIZONTAL, 5);
   headBoxSizer->Add(staticBoxSizerStatus, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
   staticBoxSizerSettings = new wxStaticBoxSizer(wxHORIZONTAL, Panel1, _("Scan Settings"));
	FlexGridSizer1 = new wxFlexGridSizer(0, 2, 0, 0);
	lblImageThreshold = new wxStaticText(Panel1, ID_STATICTEXT7, _("Brightness Threshold: 25"),
wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT7"));
   FlexGridSizer1->Add(lblImageThreshold, 1, wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
   sliderImageThreshold = new wxSlider(Panel1, ID_SLIDER3, 25, 5, 50, wxDefaultPosition,
wxSize(158,30), 0, wxDefaultValidator, _T("ID_SLIDER3"));
```

```
   FlexGridSizer1->Add(sliderImageThreshold, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
   lblBrightnessFilter = new wxStaticText(Panel1, ID_STATICTEXT5, _("Brightness Filter: 0.80"),
wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT5"));
   FlexGridSizer1->Add(lblBrightnessFilter, 1, wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
   sliderBrightnessFilter = new wxSlider(Panel1, ID_SLIDER1, 80, 0, 100, wxDefaultPosition,
wxSize(158,35), 0, wxDefaultValidator, _T("ID_SLIDER1"));
   FlexGridSizer1->Add(sliderBrightnessFilter, 1,
wxALL|wxEXPAND|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
   staticBoxSizerSettings->Add(FlexGridSizer1, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
   headBoxSizer->Add(staticBoxSizerSettings, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	StaticText2 = new wxStaticText(Panel1, ID_STATICTEXT6, _("Brightness Threshold controls how
bright the laser must be in the image for it to be detected.  Low values will be more forgiving of light
lasers or dark objects, but tend to produce more noise.\n\nBrightness Filter controls how bright pixels
must appear relative to other laser pixels.  Low values are more forgiving but tend to produce more
noise."), wxDefaultPosition, wxSize(319,170), 0, _T("ID_STATICTEXT6"));
   headBoxSizer->Add(StaticText2, 1, wxALL|wxEXPAND|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL,
5);
	BoxSizer2 = new wxBoxSizer(wxHORIZONTAL);
	butCapture = new wxButton(Panel1, ID_BUTTON3, _("Start Scan"), wxDefaultPosition,
wxSize(150,32), 0, wxDefaultValidator, _T("ID_BUTTON3"));
   butCapture->Disable();
   BoxSizer2->Add(butCapture, 1, wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL,
5);
	butDoneScanning = new wxButton(Panel1, ID_BUTTON1, _("Done Scanning"),
wxDefaultPosition, wxSize(150,32), 0, wxDefaultValidator, _T("ID_BUTTON1"));
   butDoneScanning->Disable();
   BoxSizer2->Add(butDoneScanning, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
   headBoxSizer->Add(BoxSizer2, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
   topBoxSizer->Add(headBoxSizer, 0, wxALL|wxALIGN_TOP|wxALIGN_CENTER_HORIZONTAL, 5);
   BoxSizer1->Add(topBoxSizer, 0,
wxALL|wxEXPAND|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	StaticLine1 = new wxStaticLine(Panel1, ID_STATICLINE1, wxDefaultPosition, wxSize(10,-1),
wxLI_HORIZONTAL, _T("ID_STATICLINE1"));
   BoxSizer1->Add(StaticLine1, 0,
wxALL|wxEXPAND|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	txtLog = new wxTextCtrl(Panel1, ID_TEXTCTRL1, _("------------------ Quick Tips ------------------\nThe
top 25 pixels (above the green line) are reserved for a flat surface.  If the entire top of the image is not
looking at a flat surface, the scan will not work well.\n\nTry not to move the camera or objects during
the scan.  The system uses an image-difference technique which means that it is sensitive to any
changes in the image.\n\n------------------ Initializing ------------------"), wxDefaultPosition, wxSize(999,133),
wxTE_MULTILINE|wxTE_READONLY|wxTE_WORDWRAP, wxDefaultValidator, _T("ID_TEXTCTRL1"));
   txtLog->SetMinSize(wxSize(390,154));
   BoxSizer1->Add(txtLog, 1,
wxALL|wxEXPAND|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
          Panel1->SetSizer(BoxSizer1);
   BoxSizer1->Fit(Panel1);
   BoxSizer1->SetSizeHints(Panel1);
          MenuBar1 = new wxMenuBar();
          Menu1 = new wxMenu();
          MenuItem1 = new wxMenuItem(Menu1, idMenuQuit, _("Quit\tAlt-F4"), _("Quit the
application"), wxITEM_NORMAL);
   Menu1->Append(MenuItem1);
   MenuBar1->Append(Menu1, _("&File"));
          Menu3 = new wxMenu();
          menuChangeCamera = new wxMenuItem(Menu3, idMenuChangeCamera, _("Change Camera"),
_("Change the camera if you have more than one (requires restart)"), wxITEM_NORMAL);
   Menu3->Append(menuChangeCamera);
   MenuBar1->Append(Menu3, _("Tools"));
          Menu2 = new wxMenu();
          MenuItem2 = new wxMenuItem(Menu2, idMenuAbout, _("About\tF1"), _("Show info about this
application"), wxITEM_NORMAL);
          Menu2->Append(MenuItem2);
   MenuBar1->Append(Menu2, _("Help"));
   SetMenuBar(MenuBar1);
          StatusBar1 = new wxStatusBar(this, ID_STATUSBAR1, 0, _T("ID_STATUSBAR1"));
          int __wxStatusBarWidths_1[1] = { -1 };
          int __wxStatusBarStyles_1[1] = { wxSB_NORMAL };
          StatusBar1->SetFieldsCount(1,__wxStatusBarWidths_1);
   StatusBar1->SetStatusStyles(1,__wxStatusBarStyles_1);
   SetStatusBar(StatusBar1);


Connect(ID_BUTTON4,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&ActiveStereoFr
ame::OnButCameraConnectClick);

Connect(ID_SLIDER3,wxEVT_SCROLL_THUMBTRACK,(wxObjectEventFunction)&ActiveStereoFrame::OnSl
iderImageThresholdCmdScroll);

Connect(ID_SLIDER3,wxEVT_SCROLL_CHANGED,(wxObjectEventFunction)&ActiveStereoFrame::OnSlider
ImageThresholdCmdScroll);

Connect(ID_SLIDER1,wxEVT_SCROLL_THUMBTRACK,(wxObjectEventFunction)&ActiveStereoFrame::OnSl
iderBrightnessFilterCmdScrollThumbTrack);

Connect(ID_SLIDER1,wxEVT_SCROLL_CHANGED,(wxObjectEventFunction)&ActiveStereoFrame::OnSlider
BrightnessFilterCmdScrollThumbTrack);

Connect(ID_BUTTON3,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&ActiveStereoFr
ame::OnButCaptureClick);
```

```
Connect(ID_BUTTON1,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&ActiveStereoFr
ame::OnButDoneScanningClick);

Connect(idMenuQuit,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&ActiveStereoFra
me::OnQuit);

Connect(idMenuChangeCamera,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&Active
StereoFrame::OnMenuChangeCameraSelected);

Connect(idMenuAbout,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&ActiveStereoFr
ame::OnAbout);
        //*)

        // frame icon
        #if defined(__WXMSW__)
        {
        wxIcon FrameIcon(wxT("laserIcon.ico"), wxBITMAP_TYPE_ICO);
       SetIcon(FrameIcon);
        }
        #elif defined(__UNIX__)
        {
        wxIcon FrameIcon;
    FrameIcon.CopyFromBitmap(wxBitmap(wxImage(PIXMAPS_DIR _T("/laserIcon.png"))));
    SetIcon(FrameIcon);
        }
        #endif

        // create image display window on the frame and insert it into the sizer
        m_pCamView = new CCamView( Panel1, wxPoint(150,0), wxSize(640, 480) );
  topBoxSizer->Insert(0, m_pCamView, 0, wxFIXED_MINSIZE);

        // just added the camera window, so force sizers to reset
  topBoxSizer->Layout();
        BoxSizer1->Layout();

        updateImageRunning = false;

        // init camera and laser connected variables (used for labels)
        cameraConnected = false;

        // init last USB event time (so we don't overload the PIC with USB events)
        lastUsbTime = wxDateTime::UNow();

        scanStatus = new ScanStatus();

        // read the camera preference out of the config
        wxConfig *config = new wxConfig(wxT("makerscanner"));
```

```
   config->Set(config);

           if (config->Read(wxT("CameraNum"), &cameraNum))
           {
           // read the value successfully

           // check for the -1 (first camera detected) case
           if (cameraNum == -1)
           {
           cameraNum = CV_CAP_ANY;
           }
           } else {
           cameraNum = CV_CAP_ANY;
           }

           //Start up the camera and look at the image
           cam = new Cameras(txtLog, this, scanStatus, cameraNum);  // create the camera object

   framesSinceLastFpsUpdate = 0;

   timeOfLastFpsUpdate = wxDateTime::UNow();

           //cvNamedWindow( "win1", CV_WINDOW_AUTOSIZE );
}

// Destructor
ActiveStereoFrame::~ActiveStereoFrame()
{
           // make sure to delete cam before m_pCamView otherwise
           // you might get a crash on exit in Windows OS (most likley
           // when you have a UpdateImage event after the viewer is gone)
           if (cam)
           {
           delete cam;
           }
           if (m_pCamView)
           {
           delete m_pCamView;
           }
           if (scanStatus)
           {
           delete scanStatus;
           }
           if (config)
           {
           delete config;
           }
}
```

```
// On thread destruction
void ActiveStereoFrame::OnQuit(wxCommandEvent& event)
{
  Close();
}

// On about, show a dialog with name, email, etc.
void ActiveStereoFrame::OnAbout(wxCommandEvent& event)
{
	wxString msg = wxT("MakerScanner v0.3.2\nhttp://makerscanner.com\n\n(C)
2009-2010\nAndrew Barry\nabarry@gmail.com");
	wxMessageBox(msg, _("MakerScanner"));
}

// Scan button clicked -- start a new scan!
void ActiveStereoFrame::OnButCaptureClick(wxCommandEvent& event)
{
	wxString directory = wxT("");

	// read the last directory used
	wxConfig *config = (wxConfig*)wxConfigBase::Get();
	if (config->Read(wxT("ScanDir"), &directory))
	{
	// check for a not valid directory
	if (wxDir::Exists(directory) == false)
	{
	directory = wxT("");
	}
	} else {
	// failed to read config
	directory = wxT("");
	}
	// ask where to save the point cloud file
	wxFileDialog dialog(this, wxT("Save pointcloud file"), directory, wxT("pointcloud.ply"),
wxT("*.ply"), wxFD_SAVE | wxFD_OVERWRITE_PROMPT);
	if (dialog.ShowModal() != wxID_OK)
	{
	// user clicked cancel
	return;
	}

	// get the filename and path as a string
	pointcloudFilename = dialog.GetPath();

	wxFileName dir(pointcloudFilename);
	if (dir.IsOk() == true)
	{
```

```
            // save the new selected directory
       config->Write(wxT("ScanDir"), dir.GetPath());

            }

   // clear the point cloud file
            wxFFile file(pointcloudFilename, wxT("w"));
            if (file.IsOpened())
            {
       txtLog->AppendText(wxT("\nWriting point cloud file:\n\t"));
       txtLog->AppendText(pointcloudFilename);
            file.Close();
            } else {
       txtLog->AppendText(wxT("\nPoint cloud file did not open!"));
            return;
            }

            // Disable buttons/sliders during scanning
   SetGUIStateDuringScan(true);

            if (!cam)
            {
            cam = new Cameras(txtLog, this, scanStatus, cameraNum);  // create the camera object if it
doesn't already exist
            }

   cam->SetThresholdPixelValue(sliderImageThreshold->GetValue());
   cam->SetBrightnessFilterValue(float(sliderBrightnessFilter->GetValue()) / 100.0);

            // start scanning!
   cam->StartScan();
   butDoneScanning->SetFocus();
}

// Catch an update image event and display the new image
void ActiveStereoFrame::UpdateImage(wxCommandEvent &event)
{

            // image is shipped as a pointer in the event
            // cast to IplImage (must release once finished)
            IplImage *img = (IplImage*)event.GetClientData();

            if (updateImageRunning == true)
            {
       txtLog->AppendText(wxT("\nUpdate image was already running"));
       cvReleaseImage(&img);
            return;
            }
            updateImageRunning = true;

            // camera is connected -- update if it isn't
            if (cameraConnected != true)
            {
        cameraConnected = true;
        lblCameraConnected->SetLabel(wxT("Connected"));

            // reset layouts now that the length of the label in lblCameraConnected has changed
        flexGridStatus->Layout();
        staticBoxSizerStatus->Layout();
        headBoxSizer->Layout();

        butCapture->Enable(true);
            butCapture->SetFocus();
            }

            // update image display
    m_pCamView->DrawCam(img);

            // delete the image that was copied for us to display
    cvReleaseImage(&img);

            updateImageRunning = false;

            // update the FPS display
    framesSinceLastFpsUpdate ++;


}

// catch a display text event and append the text to the terminal display
void ActiveStereoFrame::DisplayText(wxCommandEvent &event)
{
    txtLog->AppendText(event.GetString());
}

// catch a write to file event
// the string to write is in the event
// write to the point cloud file
void ActiveStereoFrame::WriteToFile(wxCommandEvent &event)
{
            wxString FilesWritten = wxT("");
            //static int i = 0;  // monitor how many times we do file writes
            //if (i==0) txtLog->AppendText(wxT("\nWriting topoint cloud file: "));
            //i++;
   //FilesWritten << i << wxT(" ");
            wxFFile file(pointcloudFilename, wxT("a"));
   file.Write(event.GetString());
            if (file.IsOpened())  txtLog->AppendText(FilesWritten);
            else txtLog->AppendText(wxT("Point cloud file did not open!"));
            file.Close();
}

// Camera connect button clicked -- attempt to connect to the camera
void ActiveStereoFrame::OnButCameraConnectClick(wxCommandEvent& event)
{
            if (cam)
            {
            delete cam;
            }
            cam = new Cameras(txtLog, this, scanStatus, cameraNum);  // create the camera object
}

// Set buttons to be enabled/disabled during (or after) a scan
void ActiveStereoFrame::SetGUIStateDuringScan(bool scanning)
{
            bool enable;
            if (scanning == true)
            {
            enable = false;
      butCapture->SetLabel(wxT("Scanning..."));

            } else {
            enable = true;
      butCapture->SetLabel(wxT("Start Scan"));
      butDoneScanning->SetLabel(wxT("Done Scanning"));
            }

            butCapture->Enable(enable);
   butDoneScanning->Enable(!enable);
   sliderImageThreshold->Enable(enable);
   sliderBrightnessFilter->Enable(enable);

   butCameraConnect->Enable(enable);

            if (scanning == false)
            {
      butCapture->SetFocus();
            }

}

// Catch an update scan event and move the progress bar
void ActiveStereoFrame::UpdateScanProgress(wxCommandEvent &event)
{
```

```
            // nothing here for now
}

// Re-enable buttons now that the scan is finished
void ActiveStereoFrame::ScanFinished(wxCommandEvent &event)
{
   SetGUIStateDuringScan(false);
}

// Done Scanning button event handler
void ActiveStereoFrame::OnButDoneScanningClick(wxCommandEvent& event)
{
            // stop the scan by telling scanStatus to stop scanning.
            // this will let the scanning thread finish it's current tasks
            // once done, the thread will send a scan finished event.
   scanStatus->SetScanning(false);
   butDoneScanning->SetLabel(wxT("Finishing Scan..."));
   butDoneScanning->Enable(false);
}

// Update the image threshold slider label on slider move
void ActiveStereoFrame::OnSliderImageThresholdCmdScroll(wxScrollEvent& event)
{
            // update the label
            wxString str = wxT("Brightness Threshold: ");
            str << sliderImageThreshold->GetValue();
   lblImageThreshold->SetLabel(str);
}

void ActiveStereoFrame::OnSliderBrightnessFilterCmdScrollThumbTrack(wxScrollEvent& event)
{
            // update the label
            wxString str = wxT("Brightness Filter: ");
            wxString numstr;
   numstr.Printf(wxT("%.2f"), float(sliderBrightnessFilter->GetValue()) / 100.0);
   lblBrightnessFilter->SetLabel(str + numstr);
}

void ActiveStereoFrame::OnMenuChangeCameraSelected(wxCommandEvent& event)
{
            // open the camera number dialog
   wxNumberEntryDialog dialog (this, wxT("Cameras are numbered in the order they are connected
in.\n\nSet the value to -1 to select the first valid camera detected."), wxT("New camera number:"),
wxT("MakerScanner - Change Camera"), cameraNum, -1, 10);

            long cameraNumNew;

            if (dialog.ShowModal() == wxID_OK)
            {
            cameraNumNew = dialog.GetValue();

            wxConfig *config = (wxConfig*)wxConfigBase::Get();

       config->Write(wxT("CameraNum"), cameraNumNew);

       wxMessageBox(wxT("You must restart MakerScanner for your changes to take effect."),
          wxT("MakerScanner - Settings Changed"));
            }
}

void ActiveStereoFrame::UpdateFps(wxUpdateUIEvent &event)
{
            wxDateTime now = wxDateTime::UNow();

            wxTimeSpan timeDiff = now.Subtract(timeOfLastFpsUpdate);

            if (timeDiff.GetMilliseconds() > 100)
            {
            // compute a new FPS
            double timeDiffNum = timeDiff.GetMilliseconds().ToDouble() / 1000.0;

            double fps = framesSinceLastFpsUpdate / timeDiffNum;

       framesSinceLastFpsUpdate = 0;

       timeOfLastFpsUpdate = now;

            wxString fpsString = wxString::Format(wxT("%4.1f"), fps);

       lblFPS->SetLabel(fpsString);

            // disable things if the fps is too low
            if (fps < 0.5)
            {
          cameraConnected = false;
          lblCameraConnected->SetLabel(wxT("Disconnected"));

            // reset layouts now that the length of the label in lblCameraConnected has changed
          flexGridStatus->Layout();
          staticBoxSizerStatus->Layout();
          headBoxSizer->Layout();

          butCapture->Enable(false);
            }
            }
```

```
}

/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
* Camera.cpp
*/

#include "Cameras.h"

// Init camera capture and get ready to start the scanning thread
Cameras::Cameras(wxTextCtrl *pMemo, wxFrame *windowIn, ScanStatus *scanStatusIn, int
cameraNumIn)
{
	window = windowIn;
	m_MyCapture = NULL;
	m_LastFrame = NULL;
	m_pMemo = pMemo;
	m_CamPaused = false;
	m_CamPlaying = false;
	cameraNum = cameraNumIn;
//	m_CamStopped = false;
	distanceToReferenceWall = -1;

	scanStatus = scanStatusIn;

	myScanThread = NULL;

	noLaserImage = NULL;
	laserCenteredImage = NULL;

	captureThread = NULL;

	thresholdPixelValue = 25;
	brightnessFilterValue = 0.8;

	// filename format for the first frame from camera 0 is MyFrame0.bmp
	// the 12th frame is MyFrame12.bmp
	m_filename =  wxT("MyFrame");   // This is the base name, integers will be appended for each
frame captured
	m_fileformat = wxT(".bmp");			// can choose bmp, jpg, etc for saving images
	m_LastCapturedFrameFilename = wxT("");
	m_FilenameIndex = wxT("");
	m_NumberOfCapturedFrames = 0;

	//cvNamedWindow("My Camera", CV_WINDOW_AUTOSIZE);

	if (InitializeCamera() == true)
	{

		captureThread = new CaptureThread(window, m_MyCapture);
		if ( captureThread->Create() != wxTHREAD_NO_ERROR )
		{
			m_pMemo->AppendText(wxT("\nFailed to create capture thread."));
		} else {
			captureThread->Run();
			captureThread->SetCapture(PREVIEW);
		}
	}
}

// kill the thread and release the camera
Cameras::~Cameras()
{
	StopCaptureThread();

	if (noLaserImage)
	{
		cvReleaseImage(&noLaserImage);
	}

	if (laserCenteredImage)
	{
		cvReleaseImage(&laserCenteredImage);
	}

	if (m_MyCapture) cvReleaseCapture(&m_MyCapture);

}

// Attempt to connect to the camera and grab a frame
bool Cameras::InitializeCamera()
```

```
{
	//m_MyCapture = cvCreateCameraCapture(CV_CAP_ANY);
	m_MyCapture = cvCaptureFromCAM(cameraNum);
	if (!m_MyCapture)
	{
		m_pMemo->AppendText(wxT("\nFailed to connect to camera."));
		return false;
	}
	cvQueryFrame(m_MyCapture); // this call is necessary to get correct capture properties

	// TODO: These resolution selection calls don't appear to work right now.
	//cvSetCaptureProperty(m_MyCapture, CV_CAP_PROP_FRAME_WIDTH, 1280);
	//cvSetCaptureProperty(m_MyCapture, CV_CAP_PROP_FRAME_HEIGHT, 960);
	m_FrameHeight   = (int) cvGetCaptureProperty(m_MyCapture, CV_CAP_PROP_FRAME_HEIGHT);
	m_FrameWidth = (int) cvGetCaptureProperty(m_MyCapture, CV_CAP_PROP_FRAME_WIDTH);
	//m_FrameRate		= (int) cvGetCaptureProperty(m_MyCapture, CV_CAP_PROP_FPS);
	wxString TempString = wxT("\nCamera online at ");
	TempString << m_FrameWidth << wxT("x") <<	m_FrameHeight << wxT(".");
	m_pMemo->AppendText(TempString);

	FrameGrab();

	return true;
}

// Determine if we are OK to capture from the camera
bool Cameras::CaptureExists()
{
	if (!m_MyCapture)
	{
		m_pMemo->AppendText(wxT("\nCamera not initialized."));
		return false;
	}
	return true;
}

// Grab a frame (note that this is NOT the same as the FrameGrab in ScanThread.cpp)

// TODO: remove this
IplImage *Cameras::FrameGrab()
{
	if (!CaptureExists()) return NULL;
  for (int i=0; i < 8; i++) cvGrabFrame(m_MyCapture); // it takes a few images to get to the newest one
  m_LastFrame = cvRetrieveFrame(m_MyCapture);
  //cvShowImage("My Camera", m_LastFrame);
	return m_LastFrame;

}
```

```
// Save an image
// TODO: remove this
void Cameras::SaveSingleFrame()
{
	if (!CaptureExists()) return;
	m_NumberOfCapturedFrames++;
	cvSaveImage(GetLastCapturedFrameFilename().char_str(), m_LastFrame);
	//m_pMemo->AppendText(wxChar((char*)GetLastCapturedFrameFilename().c_str()));
	//cvSaveImage("testImage.bmp", m_LastFrame);
}

// TODO: remove this
wxString Cameras::GetLastCapturedFrameFilename()
{
	m_LastCapturedFrameFilename = m_filename;   // basic filename
	m_LastCapturedFrameFilename << m_NumberOfCapturedFrames;
	m_LastCapturedFrameFilename.Append(m_fileformat);
		m_pMemo->AppendText(m_LastCapturedFrameFilename);

	return m_LastCapturedFrameFilename;
}

// Start the scan thread!
void Cameras::StartScan()
{
	if (GetInitialData() == true)
	{
		myScanThread = new ScanThread(window, captureThread, scanStatus, noLaserImage, 
laserCenteredImage, distanceToReferenceWall);
		if ( myScanThread->Create() != wxTHREAD_NO_ERROR )
		{
			m_pMemo->AppendText(wxT("\nFailed to create scan thread."));
		} else {
			scanStatus->SetScanning(true);
			myScanThread->SetThresholdPixelValue(thresholdPixelValue);
			myScanThread->SetBrightnessThreshold(brightnessFilterValue);
			myScanThread->Run();
		}
	}
}

// Shutdown the thread
void Cameras::StopCaptureThread()
{
	if (captureThread)
	{
		captureThread->SetCapture(STOP);
```

```
			captureThread->Wait();
			captureThread = NULL;
		}
}

bool Cameras::GetInitialData()
{
		// pop up the dialog box to ask the user to center the laser or give us the distance
		// to the reference wall.

		DistanceToReferenceDialog distDiag(captureThread, window, wxID_ANY);

		int returnNum = distDiag.ShowModal();

		if (returnNum == wxID_OK)
		{
			distanceToReferenceWall = distDiag.GetWallDistance();
			noLaserImage = distDiag.GetNoLaserImage();

		} else if (returnNum == USER_CENTERED_LASER)
		{

			noLaserImage = distDiag.GetNoLaserImage();
			laserCenteredImage = distDiag.GetLaserCenteredImage();


		} else {
			// cancelled
			return false;
		}
		return true;
}

////////////////////////////////////////////////////////////////////
// Name:       implementation of the CCamView class
// File:       camview.cpp
// Purpose:    eye/camera view/GUI system control methods
//
// Created by:  Larry Lart on 22-Feb-2006
// Updated by:  Andrew Barry Jan 2010
//
// Copyright:   (c) 2006 Larry Lart
////////////////////////////////////////////////////////////////////


// on windows, you must include the wx headers before
// other headers - Andy
#include <wx/image.h>
```

```
// main header -- this has wx headers in it so make
// sure to include it before other headers - Andy
#include "camview.h"


// system header
#include <math.h>
#include <stdio.h>
#include "cv.h"
#include "highgui.h"


#ifdef __GNUG__
#pragma implementation
#pragma interface
#endif

// implement message map
BEGIN_EVENT_TABLE(CCamView, wxWindow)
	EVT_PAINT( CCamView::OnPaint )
	EVT_SIZE( CCamView::OnSize )
END_EVENT_TABLE()

////////////////////////////////////////////////////////////////////
// Method:  Constructor
// Class:   CCamView
// Purose:  build my CCamView object
// Input:   nothing
// Output:  nothing
////////////////////////////////////////////////////////////////////
CCamView::CCamView( wxWindow *frame, const wxPoint& pos, const wxSize& size ):
        wxWindow(frame, -1, pos, size, wxSIMPLE_BORDER )
{
	//m_pCamera = NULL;

	// set my canvas width/height
	m_nWidth = size.GetWidth( );
	m_nHeight = size.GetHeight( );

	m_bDrawing = false;

	m_bNewImage = 0;

	m_pBitmap = NULL;

}
```

```
////////////////////////////////////////////////////////////////////
// Method:  Destructor
// Class:   CCamView
// Purose:  destroy my object
// Input:   nothing
// Output:  nothing
////////////////////////////////////////////////////////////////////
CCamView::~CCamView( )
{
	//m_pCamera = NULL;
}

////////////////////////////////////////////////////////////////////
// Method:  Is Capture Enabled
// Class:   CCamView
// Purose:  check if camera is initialized
// Input:   nothing
// Output:  bool yes/no
////////////////////////////////////////////////////////////////////
bool CCamView::IsCaptureEnabled( )
{
//  return( m_pCamera->IsInitialized( ) );
	return( 1 );
}

////////////////////////////////////////////////////////////////////
// Method:  OnPaint
// Class:   CCamView
// Purose:  on paint event
// Input:   reference to paint event
// Output:  nothing
////////////////////////////////////////////////////////////////////
void CCamView::OnPaint( wxPaintEvent& event )
{
	wxPaintDC dc(this);
	Draw( dc );
}

////////////////////////////////////////////////////////////////////
// Method:  Draw
// Class:   CCamView
// Purose:  camera drawing
// Input:   reference to dc
// Output:  nothing
////////////////////////////////////////////////////////////////////
void CCamView::Draw( wxDC& dc )
{
```

```
));


            // check if dc available
            if( !dc.IsOk( ) || m_bDrawing == true ){ return; }

            m_bDrawing = true;

            int x,y,w,h;
        dc.GetClippingBox( &x, &y, &w, &h );
            // if there is a new image to draw
            if( m_bNewImage )
            {
          dc.DrawBitmap( *m_pBitmap, x, y );
          m_bNewImage = false;
            } else
            {
                // draw inter frame ?
            }

            m_bDrawing = false;

            return;
}

////////////////////////////////////////////////////////////////////
// Method:  OnDraw
// Class:   CCamView
// Purose:  CCamView drawing
// Input:   nothing
// Output:  nothing
////////////////////////////////////////////////////////////////////
void CCamView::DrawCam( IplImage* pImg )
{
//  return;
            if( m_bDrawing ) return;
            m_bDrawing = true;
            // if there was an image then we need to update view
   if( pImg )
            {
            // copy the image (will be deleted after display)
            IplImage *pDstImg = pImg;//cvCloneImage(pImg);

            int nCamWidth = pImg->width;//m_pCamera->m_nWidth;
            int nCamHeight = pImg->height;//m_pCamera->m_nHeight;


            // draw a vertical line through the center of the image
        cvLine(pDstImg, cvPoint(nCamWidth/2, 0), cvPoint(nCamWidth/2, nCamHeight), CV_RGB( 0,255,0
));
```

```
            // draw a horizontal line at pixel 25
        cvLine(pDstImg, cvPoint(0, 25), cvPoint(nCamWidth, 25), CV_RGB( 0,255,0 ));

            // draw a horizontal line through the center of the image
        //cvLine(pDstImg, cvPoint(0, nCamHeight/2), cvPoint(nCamWidth, nCamHeight/2), CV_RGB( 0,255,0
));

            // process image from opencv to wxwidgets
            unsigned char *rawData;
            // draw my stuff to output canvas
            CvSize roiSize;
            int step = 0;

            // get raw data from ipl image
            cvGetRawData( pDstImg, &rawData, &step, &roiSize );

            // convert data from raw image to wxImg


            wxImage *pWxImg = new wxImage( nCamWidth, nCamHeight, rawData, TRUE );

            // convert to bitmap to be used by the window to draw

            if (m_pBitmap)
            {
            delete m_pBitmap;
            }

            m_pBitmap = new wxBitmap( pWxImg->Scale(m_nWidth, m_nHeight) );

            m_bNewImage = true;
            m_bDrawing = false;

            Refresh( FALSE );

            //Update( );
            delete pWxImg;

        //cvReleaseImage( &pDstImg );


            }

}

//////////////////////////////////////////////////////////////////
// Method:  CheckUpdate
// Class:   CCamView
// Purose:  CHeck for updates
// Input:   reference to size event
// Output:  nothing
////////////////////////////////////////////////////////////////////
void CCamView::CheckUpdate()
{
        Update( );
}

////////////////////////////////////////////////////////////////////
// Method:  OnSize
// Class:   CCamView
// Purose:  adjust on windows resize
// Input:   reference to size event
// Output:  nothing
////////////////////////////////////////////////////////////////////
void CCamView::OnSize( wxSizeEvent& even )
{
        int nWidth = even.GetSize().GetWidth();
        int nHeight = even.GetSize().GetHeight();

        m_nWidth = nWidth;
        m_nHeight = nHeight;

}

/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/


#include "CaptureThread.h"

DEFINE_EVENT_TYPE(IMAGE_UPDATE_EVENT)
```

```
CaptureThread::CaptureThread(wxFrame *windowIn, CvCapture *captureIn) :
wxThread(wxTHREAD_JOINABLE)
{
            capturing = IDLE;
            window = windowIn;
            cvCapture = captureIn;
}

// called on thread quit -- free all memory
void CaptureThread::OnExit()
{

}

// Called when thread is started
void* CaptureThread::Entry()
{
            while (true)
            {
            // check to see if the thread should exit
            if (TestDestroy() == true)
            {
            break;
            }

            if (capturing == CAPTURE)
            {
            // get a new image
          CaptureFrame();
            } else if (capturing == PREVIEW)
            {

            // get a new image and show it on screen
          CaptureFrame();
          SendFrame(imageQueue.back());
            } else if (capturing == IDLE)
            {
            Sleep(10);
            } else if (capturing == STOP)
            {
            break;
            }

            Yield();
            }

            return NULL;
```

```
}

void CaptureThread::CaptureFrame()
{
            if (!cvCapture){
              //fail
            return;
            }

            if (imageQueue.size() > 100)
            {
            // stack too big, throw out some data
       imageQueue.pop();
            }

            for (int i=0; i < 1; i++) cvGrabFrame(cvCapture); // it takes a few images to get to the newest one
            IplImage* lastFrame = cvRetrieveFrame(cvCapture);
            // cvShowImage("My Camera", LastFrame);
   imageQueue.push(lastFrame);

}

IplImage* CaptureThread::Pop()
{
            if (imageQueue.size() <= 0)
            {
       CaptureFrame();
            }

            IplImage *image = imageQueue.front();

            if (imageQueue.size() > 1)
            {
       imageQueue.pop();
            }

            return image;
}

/*
* Flush the stack, allowing the user to make sure s/he gets the most
* up to date image.  Delete all images in the stack.
*/
void CaptureThread::Flush()
{
            CaptureStatus oldCap = capturing;

            capturing = IDLE;
            while (imageQueue.size() > 0)
            {
        imageQueue.pop();

            // since you should never release an image gotten by cvRetrieveFrame,
            // we don't need to release images here.
            }

            capturing = oldCap;
}

// Display the given image on the frame
// Copies the image so it is safe to change it after the function call
void CaptureThread::SendFrame(IplImage *frame)
{
            if (!frame)
            {
            return;
            }

            IplImage* pDstImg;
            CvSize sz = cvSize(frame->width, frame->height);
            pDstImg = cvCreateImage(sz, 8, 3);
            cvZero(pDstImg);
            // convert the image into a 3 channel image for display on the frame
            if (frame->nChannels == 1)
            {
        //cvCvtColor(frame, pDstImg, CV_GRAY2BGR);

            // another way to convert grayscale to RGB
            cvMerge(frame, frame, frame, NULL, pDstImg);
            } else if (frame->nChannels == 3){

            // opencv stores images as BGR instead of RGB so we need to convert
        cvConvertImage(frame, pDstImg, CV_CVTIMG_SWAP_RB);

            } else {
            // we don't know how to display this image based on its number of channels

            // give up
        cvReleaseImage( &pDstImg );
            return;
            }

            wxCommandEvent event(IMAGE_UPDATE_EVENT, GetId());

            // send the image in the event
```

```
   event.SetClientData(pDstImg);

		// Send the event to the frame!
   window->GetEventHandler()->AddPendingEvent(event);
}

#include "DistanceToReferenceDialog.h"

//(*InternalHeaders(DistanceToReferenceDialog)
#include <wx/string.h>
#include <wx/intl.h>
//*)

//(*IdInit(DistanceToReferenceDialog)
const long DistanceToReferenceDialog::ID_STATICTEXT1 = wxNewId();
const long DistanceToReferenceDialog::ID_STATICLINE1 = wxNewId();
const long DistanceToReferenceDialog::ID_STATICTEXT4 = wxNewId();
const long DistanceToReferenceDialog::ID_BUTTON1 = wxNewId();
const long DistanceToReferenceDialog::ID_STATICTEXT5 = wxNewId();
const long DistanceToReferenceDialog::ID_BUTTON2 = wxNewId();
const long DistanceToReferenceDialog::ID_STATICTEXT6 = wxNewId();
const long DistanceToReferenceDialog::ID_STATICTEXT7 = wxNewId();
const long DistanceToReferenceDialog::ID_SPINCTRL1 = wxNewId();
const long DistanceToReferenceDialog::ID_PANEL1 = wxNewId();
//*)

BEGIN_EVENT_TABLE(DistanceToReferenceDialog,wxDialog)
	//(*EventTable(DistanceToReferenceDialog)
	//*)
END_EVENT_TABLE()

DistanceToReferenceDialog::DistanceToReferenceDialog(CaptureThread *captureThreadIn, wxWindow*
parent,wxWindowID id,const wxPoint& pos,const wxSize& size)
{
	//(*Initialize(DistanceToReferenceDialog)
	wxFlexGridSizer* FlexGridSizer1;
	wxBoxSizer* BoxSizer2;
	wxBoxSizer* BoxSizer1;
	wxBoxSizer* BoxSizer5;

	Create(parent, wxID_ANY, _("Distance to Reference Surface"), wxDefaultPosition,
wxDefaultSize, wxDEFAULT_DIALOG_STYLE, _T("wxID_ANY"));
	SetClientSize(wxSize(446,289));
	BoxSizer1 = new wxBoxSizer(wxHORIZONTAL);
	Panel1 = new wxPanel(this, ID_PANEL1, wxDefaultPosition, wxDefaultSize, wxTAB_TRAVERSAL,
_T("ID_PANEL1"));
	BoxSizer2 = new wxBoxSizer(wxVERTICAL);
	BoxSizer5 = new wxBoxSizer(wxHORIZONTAL);
	StaticText1 = new wxStaticText(Panel1, ID_STATICTEXT1, _("Before we start, we need to know a
few things:\n\n1) what your object looks like without the laser.\n2) how far away your flat reference
surface is."), wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT1"));
	BoxSizer5->Add(StaticText1, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	BoxSizer2->Add(BoxSizer5, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	StaticLine1 = new wxStaticLine(Panel1, ID_STATICLINE1, wxDefaultPosition, wxSize(10,-1),
wxLI_HORIZONTAL, _T("ID_STATICLINE1"));
	BoxSizer2->Add(StaticLine1, 0,
wxALL|wxEXPAND|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	FlexGridSizer1 = new wxFlexGridSizer(0, 2, 0, 0);
	lblCoverLaser = new wxStaticText(Panel1, ID_STATICTEXT4, _("1) Cover the laser"),
wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT4"));
	FlexGridSizer1->Add(lblCoverLaser, 1, wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
	butLaserCovered = new wxButton(Panel1, ID_BUTTON1, _("Done"), wxDefaultPosition,
wxDefaultSize, 0, wxDefaultValidator, _T("ID_BUTTON1"));
	FlexGridSizer1->Add(butLaserCovered, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	lblCenterLaser = new wxStaticText(Panel1, ID_STATICTEXT5, _("2) Center the laser"),
wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT5"));
	lblCenterLaser->Disable();
	FlexGridSizer1->Add(lblCenterLaser, 1, wxALL|wxALIGN_LEFT|wxALIGN_CENTER_VERTICAL, 5);
	butLaserCentered = new wxButton(Panel1, ID_BUTTON2, _("Done"), wxDefaultPosition,
wxDefaultSize, 0, wxDefaultValidator, _T("ID_BUTTON2"));
	butLaserCentered->Disable();
	FlexGridSizer1->Add(butLaserCentered, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	lblOr = new wxStaticText(Panel1, ID_STATICTEXT6, _("or"), wxDefaultPosition, wxSize(45,17), 0,
_T("ID_STATICTEXT6"));
	lblOr->Disable();
	FlexGridSizer1->Add(lblOr, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	FlexGridSizer1->Add(-1,-1,1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	lblEnterDistance = new wxStaticText(Panel1, ID_STATICTEXT7, _("Enter the distance to the\nflat
reference wall (cm):"), wxDefaultPosition, wxDefaultSize, 0, _T("ID_STATICTEXT7"));
	lblEnterDistance->Disable();
	FlexGridSizer1->Add(lblEnterDistance, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	spinDistance = new wxSpinCtrl(Panel1, ID_SPINCTRL1, _T("50"), wxDefaultPosition,
wxSize(79,27), 0, 0, 1000, 50, _T("ID_SPINCTRL1"));
	spinDistance->SetValue(_T("50"));
	spinDistance->Disable();
	FlexGridSizer1->Add(spinDistance, 0,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	FlexGridSizer1->Add(-1,-1,1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
```

```
	butDistanceEntered = new wxButton(Panel1, wxID_OK, _("Done"), wxDefaultPosition,
wxDefaultSize, 0, wxDefaultValidator, _T("wxID_OK"));
	butDistanceEntered->Disable();
	FlexGridSizer1->Add(butDistanceEntered, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	BoxSizer2->Add(FlexGridSizer1, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	Panel1->SetSizer(BoxSizer2);
	BoxSizer2->Fit(Panel1);
	BoxSizer2->SetSizeHints(Panel1);
	BoxSizer1->Add(Panel1, 1,
wxALL|wxALIGN_CENTER_HORIZONTAL|wxALIGN_CENTER_VERTICAL, 5);
	SetSizer(BoxSizer1);
	BoxSizer1->SetSizeHints(this);


Connect(ID_BUTTON1,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&DistanceToRef
erenceDialog::OnButLaserCoveredClick);

Connect(ID_BUTTON2,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction)&DistanceToRef
erenceDialog::OnButLaserCenteredClick);
	//*)

	butLaserCovered->SetFocus();
	captureThread = captureThreadIn;
}

DistanceToReferenceDialog::~DistanceToReferenceDialog()
{
	//(*Destroy(DistanceToReferenceDialog)
	//*)
}


void DistanceToReferenceDialog::OnButLaserCenteredClick(wxCommandEvent& event)
{
	captureThread->Flush();
	IplImage *laserCenteredTemp = captureThread->Pop();

	laserCentered = cvCloneImage(laserCenteredTemp);

	EndModal(USER_CENTERED_LASER);
}


bool DistanceToReferenceDialog::TransferDataFromWindow()
{
	distance = spinDistance->GetValue();
	return true;
}

void DistanceToReferenceDialog::OnButLaserCoveredClick(wxCommandEvent& event)
{
	captureThread->Flush();
	IplImage *tempNoLaserImage = captureThread->Pop();

	noLaserImage = cvCloneImage(tempNoLaserImage);


	butLaserCovered->Enable(false);
	lblCoverLaser->Enable(false);

	lblCenterLaser->Enable(true);
	butLaserCentered->Enable(true);
	lblOr->Enable(true);
	lblEnterDistance->Enable(true);
	spinDistance->Enable(true);
	butDistanceEntered->Enable(true);

	butLaserCentered->SetFocus();

}

/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "PointCloud.h"

#include <wx/listimpl.cpp>
```

```
// define a linked list of PointCloudPoints
WX_DEFINE_LIST(ListOfCloudPoints);

// PointCloud holds PointCloudPoints which are individual lines in a pointcloud file
// Points are identified by height in the image and by laser position
PointCloud::PointCloud()
{
	cloudMap = new CloudMap();
	numberOfPoints = 0;
}

// Destructor -- clear the hash map and delete all PointCloudPoints
PointCloud::~PointCloud()
{

	// empty the hash map

	CloudMap::iterator it;

	for( it = cloudMap->begin(); it != cloudMap->end(); ++it )
	{
		delete it->second;
	}

	delete cloudMap;
}

// The points are uniquely identified by h and laserPos.
// Index is -- laserPos * 1000 + h yielding an set size of 255*1000 + 1000 = 256,000
// Each hash map container holds a linked list of PointCloudPoints, which can be combined in an average
// for that point
int PointCloud::GetKey(int h, int laserPos)
{
	return laserPos * 1000 + h;
}

// Helper function to add a point that creates the PointCloudPoint for you
void PointCloud::AddPoint(double dist, double theta, double phi, int r, int g, int b, int w, int h, int
laserPos)
{
	// make a new point
	PointCloudPoint *newPoint = new PointCloudPoint(dist, theta, phi, r, g, b, w, h, laserPos);
	AddPoint(newPoint);
}

// Add a point to the point cloud.  If there is not already a point at this h, and laserPos, make
// a new linked list.  Otherwise, add to the linked list that we already have for this location
void PointCloud::AddPoint(PointCloudPoint *point)
```

```
{
	// first, get the list
	ListOfCloudPoints *thisList;

	if (cloudMap->find(GetKey(point->h, point->laserPosition)) == cloudMap->end())
	{
		// this is a new point, create a new list
		thisList = new ListOfCloudPoints();
		(*cloudMap)[GetKey(point->h, point->laserPosition)] = thisList;
		numberOfPoints ++;

	} else {
		thisList = (*cloudMap)[GetKey(point->h, point->laserPosition)];
	}

	// add the point to the list

	thisList->Append(point);


}

// Return a .ply file containing the entire point cloud (large string!)
wxString PointCloud::GetPointCloudPly()
{
	wxString cloudString = wxT("");

	wxString headerString = wxT("ply");
	headerString += wxT("\nformat ascii 1.0");

	headerString += wxT("\nelement vertex ");
	headerString << numberOfPoints;

	headerString += wxT("\nproperty float x");
	headerString += wxT("\nproperty float y");
	headerString += wxT("\nproperty float z");
	headerString += wxT("\nproperty uchar diffuse_red");
	headerString += wxT("\nproperty uchar diffuse_green");
	headerString += wxT("\nproperty uchar diffuse_blue");
	headerString += wxT("\nelement face 0");
	headerString += wxT("\nproperty list uchar int vertex_indices");
	headerString += wxT("\nend_header");

	CloudMap::iterator it;
	ListOfCloudPoints *thisList;

	for( it = cloudMap->begin(); it != cloudMap->end(); ++it )
	{
```

```
                thisList = it->second;

                PointCloudPoint thisPoint = AverageList(thisList);

                cloudString += wxT("\n") + thisPoint.GetPlyString();

        }

        return headerString + cloudString;
}

// average a linked list of points to handle a case where we have more than one point per h, laserPos
PointCloudPoint PointCloud::AverageList(ListOfCloudPoints *thisList)
{
        double number = double(thisList->GetCount());

        double sumDist = 0;

        PointCloudPoint *point = NULL;

        // iterate over the list and average all values
        for ( ListOfCloudPoints::Node *node = thisList->GetFirst(); node; node = node->GetNext() )
        {

                point = node->GetData();
                sumDist += point->dist;
        }

        if (point == NULL)
        {
                return PointCloudPoint(0, 0, 0, 0, 0, 0, 0, 0, 0);
        }

        return PointCloudPoint(sumDist/number, point->theta, point->phi, point->r, point->g, point->b,
point->w, point->h, point->laserPosition);

}

/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "PointCloudPoint.h"

// copy constructor
PointCloudPoint::PointCloudPoint(PointCloudPoint *point)
{
	dist = point->dist;
	theta = point->theta;
	phi = point->phi;

	r = point->r;
	g = point->g;
	b = point->b;
	w = point->w;
	h = point->h;
	laserPosition = point->laserPosition;
}

// Generates the point cloud string for this point (one line)
wxString PointCloudPoint::GetPlyString()
{

	double pxDist = dist;


	double x, y, z;

	x = pxDist * tan(theta);
	y = pxDist;
	z = pxDist * tan(phi);

	wxString xString = wxT("");
	xString << x;

	wxString yString = wxT("");
	yString << y;

	wxString zString = wxT("");
	zString << z;

	wxString rString = wxT("");
	rString << r;

	wxString gString = wxT("");
	gString << g;

	wxString bString = wxT("");
	bString << b;

	return xString + wxT(" ") + yString + wxT(" ") + zString + wxT(" ") + rString + wxT(" ")
		+ gString + wxT(" ") + bString;

}

/*
* Copyright 2009-2010, Andrew Barry
*
* This file is part of MakerScanner.
*
* MakerScanner is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License (Version 2, June 1991) as published by
* the Free Software Foundation.
*
* MakerScanner is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/


#include "ScanStatus.h"

ScanStatus::ScanStatus()
{
	scanning = false;
}

ScanStatus::~ScanStatus()
{
}
```