Project Report 18-551, Fall 2006

Strike a Pose: The 3-D Modeler

Group 2

Steven Nielsen (snielsen@andrew.cmu.edu)

Garrett Jenkinson (wgj@andrew.cmu.edu)

Gary Garvin (ggarvin@andrew.cmu.edu)

The Problem

The production of accurate 3-dimensional models of objects and surfaces is a common problem for many engineering and artistic tasks. From computer aided design problems and biometric recognition, to video game and animated movie creation, there are many uses for three-dimensional modeling. Unfortunately for most people, the devices required to obtain these models can be quite expensive, far too much for the independent artist or hobby engineer. Without expensive laser distance measurement technology it is non-trivial to transition special information into an accurate three dimensional model in the computer, and this difficult task can be rather computationally intensive if clever tricks and algorithms are not properly employed.

The Solution

We propose a solution which will allow us to construct a cheap and accurate three dimensional scanner. We plan to keep the total cost of the project under $75, and be able to reconstruct at least a 6" by 6" by 6" object with a fair degree of detail. We have found Matlab code which served as a proof-of-concept for many of the theoretical foundations of our project, however this rather rudimentary script provided little actual code to our final implementation. In fact with the exception of the invariable trigonometric operations the code had to be completely re-written and transposed into C. The basic setup of our system will involve placing the desired object on a rotating platform, and then projecting a vertical line with a laser on the object.

By photographing the object at a near perpendicular to the laser we achieve a method of measuring the distance from the axis of rotation to the surface of the object. By rotating the object 360 degrees we obtain enough information to reconstruct the three-dimensional surface. Since our setup will not have the detailed accuracy of the more expensive laser scanners we hope to obtain the general shape and some surface details of the object. The industrial expensive solution manipulates a series of laser rangefinders with an intense rotational feedback controlling method to obtain extremely accurate depth information to analyze a stationary object. By using a camera as the only sensor and keeping the rig stationary while rotating the object to obtain directional information, we have found a clever way of turning a complicated robotic controls and problem into a cheap and efficient video processing problem.

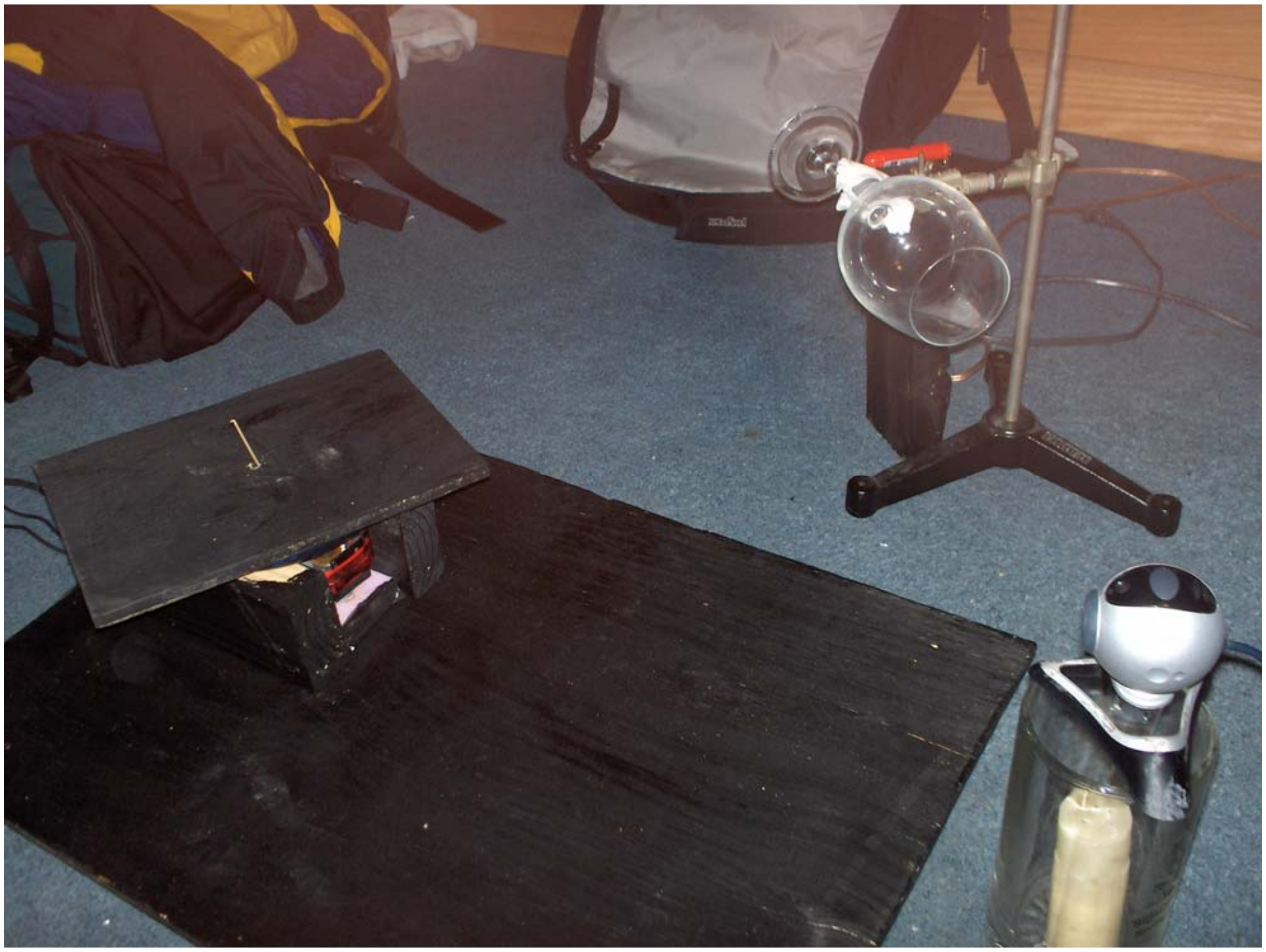
What We've Done

In order to execute this project, we first had to construct a fairly simple rig from parts which most people can obtain at a low cost (indeed, we spent no money on any parts during our construction with the exception of $3 of black paint). This involved first constructing a rotating platform on which the object to be scanned will sit, and a secondly erecting a structure that had a sturdy support for a camera and laser pointer. It was very important that neither of these objects shift position or vibrate excessively during a scan, since doing so will skew results in the reconstruction process. The rig also had to support the cylindrical lens whose task is to convert the laser pointer's singular dot into a more usable vertical laser stripe. The rotating platform can be quite simple as well and we were looking into using a record player to fulfill this requirement, but ended up constructing our own rotating platform from available materials we found for free.

We used a simple motor with low speed and high torque to rotate the object at a constant speed. Since we do not know the size and weight of all objects that we may want to scan it is important that this motor has the strength to rotate heavier loads. We initially planned to drive the rotating platform by a system of belts driven by the motor, but for the sake of simplicity and reliability we directly connected the motor to the rotating platform. The entire platform is designed to support rather heavy, uneven loads so it needs to be strong enough to maintain a level surface as the object moves about the full circle. To accommodate these uneven loads which would destabilize our platform, we dissected a plastic Frisbee to provide a track-like support system around the perimeter of the rotating platform. The plastic Frisbee is ideal because it is an inexpensive and readily available part which is already circular and adds only a reasonable amount of friction resistance to the system.

The motor choice was based on our desire for a slow rotating platform. The motor selected was able to plug directly into the wall and is geared for 8 rotations a minute. This gearing gives the motor ample amounts of torque which is ideal for larger, more massive objects. Due to the added friction and weight of objects we found that we were not getting the ideal 8 revolutions per minute. This can easily be accommodated by timing the actual rotation time, and adjusting the parameters in the GUI to access the full rotation in the AVI file. As long as we have access to a full rotation, a slower motor is no hindrance and actually provides the model with greater angular detail.

The webcam used in this project was a Logitech Quickcam Pro 4000 which was made available by Professor Casasent from previous 551 projects. We use a lower resolution video mode which takes 640 by 480 pixel video at 15 frames per second. Since a rotation of our platform takes 12 seconds to complete, we are provided 180 images corresponding to a vertical stripe of information about the object every two degrees. The Logitech webcam proved troublesome since Logitech does not provide a compatible software development kit (SDK). If this project were to be reattempted in the future, a webcam with a higher frame rate and resolution would improve upon the model reconstruction.



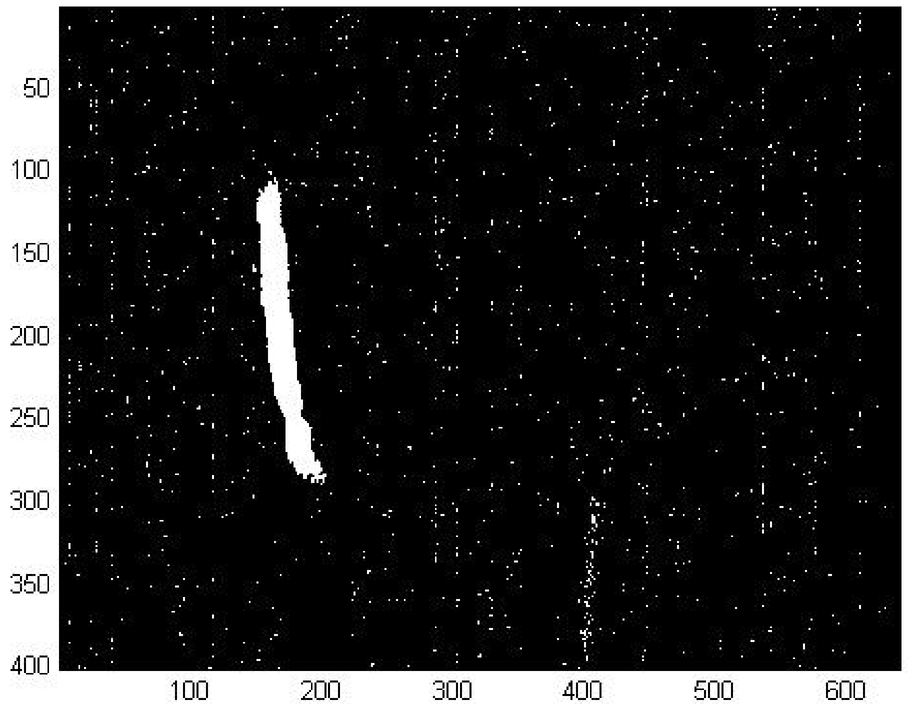
3d Laser Scanner Set-Up

For the vertical laser beam, we iterated our design multiple times in search of an ideal, cost-effective solution. We began our project using an inexpensive red keychain laser which we shined through the stem of a wine glass. The wine glass stem approximates a cylindrical lens and the quality of the glass (i.e. the lack of imperfections and bubbles) make it an ideal cheap solution to our problem. We experimented with various cylindrical acrylic dowels, but found that the scattering due to internal debris and bubbles from cheap manufacturing processes was unacceptable for our application. We also experimented with specifically designed laser pointer tips which transformed the beam into a vertical stripe. In our experience, the inexpensive red lasers at our disposal were unable to drive a sufficiently bright line through this inefficient laser pointer tip. We then borrowed a rather high quality green laser pointer which we were able to (against the manufacturer's intensions) modify by adjusting an easily accessible variable resistor. The legal laser power without a special license within the United States is 5mW, and all our red lasers complied with this requirement. However after modification of our green laser, we were able to achieve laser powers of 250+mW which is dangerous enough to be able to light a matchbook. We were definitely able to produce a sufficiently bright light stripe through the laser pointer tip, but because the green laser was not designed with that amount of power in mind, the diffraction and diffusion patterns were large and completely unacceptable. For both safety and practical reasons we have chosen a red laser of legal power in combination with a wine glass for our final, cost-effective design.

The mathematics and physics of cylindrical lens support our choice of a wine glass stem with a generic red laser pointer. There are two mathematical descriptions which explain how the wine glass stem transforms the laser pointer's beam into a vertical stripe: the Fourier transform properties of the lens, and the generalized lens focal point physics formulas. Because the cross section of the wine glass stem is constant in one direction and circular in the other (and hence cylindrical in nature) it will make no changes to a signal in the constant dimension while taking the Fourier transform of a signal in the other dimension. Since the laser pointer's input to the wine glass stem is well approximated by a Dirac delta function, we see that in the vertical direction, the Fourier transform of a Dirac pulse is a constant and in the horizontal direction, the signal is left as a delta function. Because of this, a Dirac delta function in the horizontal direction multiplied by a constant in the vertical direction corresponds to vertical line. The other, less intuitive way of viewing the wine glass-laser pointer system is through lens theory. Lens theory shows that the focal point of a cylindrical lens is at the center point of the cylinder. Because the light passes through the focal point and out the rest of the lens, the light will diverge from the focal point (and when one is infinitely far away the light beam will be an infinitely tall vertical laser stripe). Use of a lens as apposed to the vertical slit which the laser tip utilized provides the attractive property of no horizontal interference patterns (though in the vertical direction we will see that since the "delta function" produced by our laser pointer is actually more of a tight two-dimensional rectangular box function whose Fourier transform will actually be a very wide sinc function with nulls and side-lobes). Because the laser light is monochrome we note that passing the beam through a slit (as was done in the laser pointer tip) produces unwanted scattering and interference patterns as described in the wave theory of photons. We leave the interested reader to research further wave theory to better describe the negative effects of passing our beam through a vertical slit.

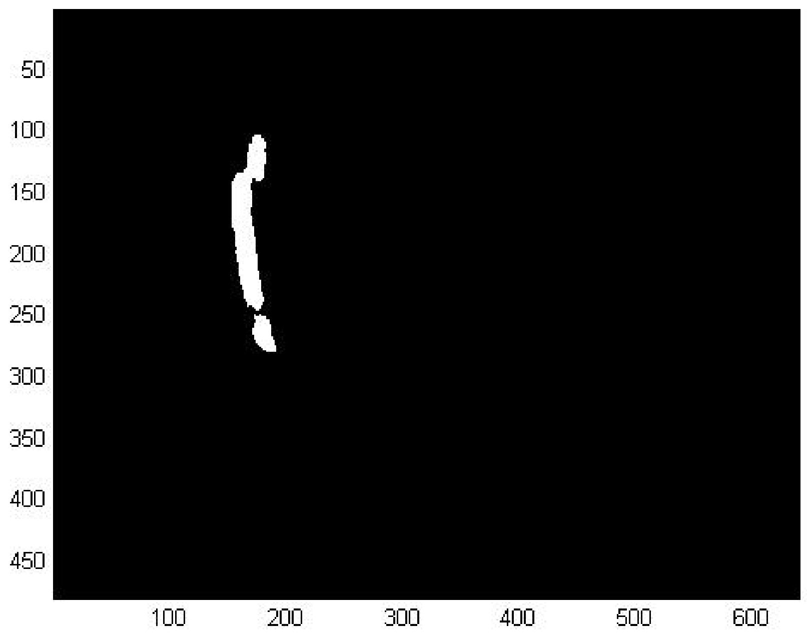
While gathering datasets, our inexpensive and rather small scale system clearly cannot bear the weight of a human subject whose face we wish to scan. We improvised our system to gather data from a seated subject in a rotating desk chair. Human error is intrinsically introduced into our system because the rotation of the chair is no longer controlled by a motor, and is instead driven by a person at as constant of a rate as possible. Empirically we have found this method does not lead to noticeable distortions as long as the rotation is kept sufficiently slow, and constant through a known angle of rotation. Future efforts may be devoted to driving the desk chair with a motor-belt gearing system, but we view our project as a proof-of-concept for these larger scale scans. We even argue that larger objects will yield more attractive results because the laser stripe halo will be proportionally smaller to the distance from the axis of rotation of the object and will better approximate a perfect, infinitely thin line.

For the software processing side of this project, we need to consider the transition of the image data from the camera to the computer and then from the computer to the DSK board and finally from the DSK board back to the computer for visualization. As done in the labs we will be paging over the image data for processing on the DSK. By using a red laser pointer and only the red layer of the color image, we are hoping to minimize not only the amount of data being transferred and processed, but also the need for noise removal after the edge detection processing. We originally had planned on using edge detection to receive a binary edge map, but found this technique to be horribly inefficient with unsatisfying results. Canny edge detection was chosen because it has been shown to be the optimal edge detection algorithm, but we found the computational intensity and quality of output from canny edge detection did not justify its use. In the less computationally intensive case, we assembled a trivial thresholding routine utilizing user determined thresholding parameters. After thresholding is complete we performed some noise reduction techniques, and once again the complexity of our algorithm was determined empirically by the presence of noise after our edge detection algorithm. In the most complex case we employed morphological processing to eliminate small noise segments and to fix breaks in our vertically projected stripe. In the more trivial case we averaged the location of the "on" bits edge horizontally. Through experimentation, we have found that the noise reduction and line continuity gained through binary morphological closing is well worth the computational cost, and that median filtering is more efficient and worthwhile than thinning for estimating the horizontal pixel location of the laser stripe.



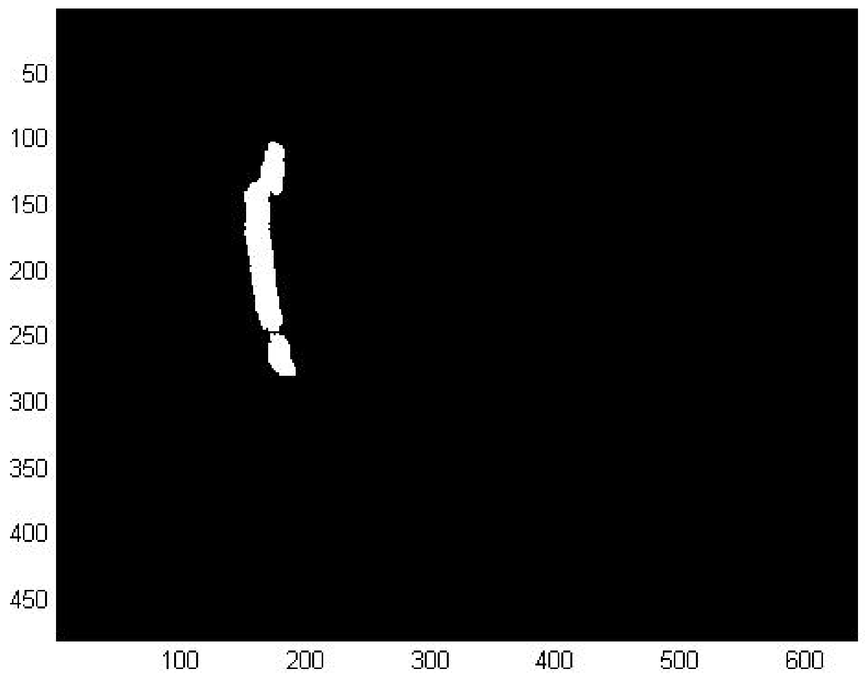
Original Image After Thresholding

The morphological operation of closing has a two-fold effect on our binary thresholded images. First it eliminates small stray noisy bits which are left behind after our simple thresholding routine. Second it fills in sufficiently small breaks in our vertical stripe, making the three dimensional reconstruction of our object more smooth with less data breaks. Morphological operations are non-linear small scale operations performed across the image, and closing involves the act of dilation followed by erosion.



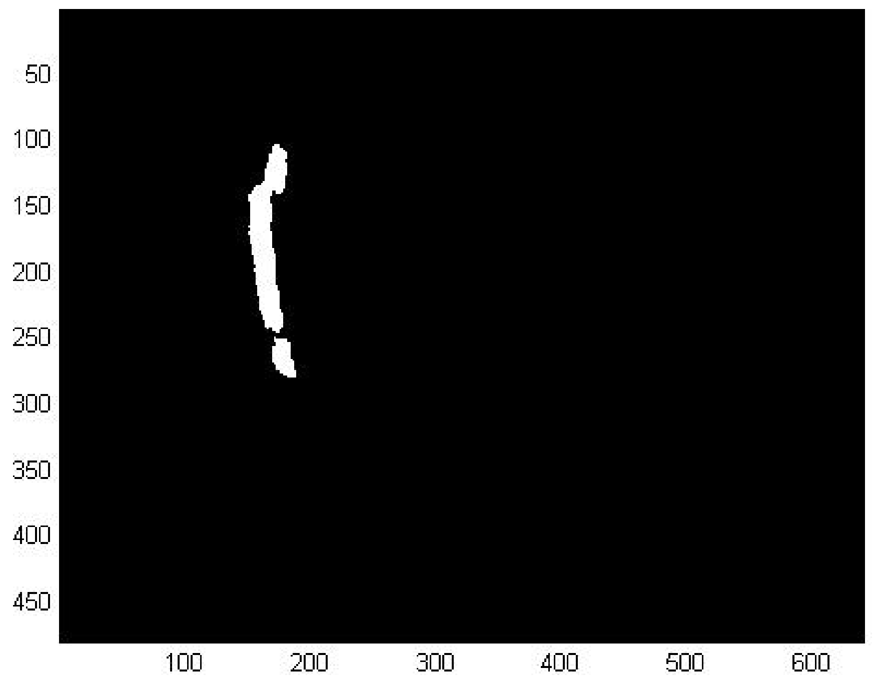
After the First Erode

Dilation, as the name implies, has the effect of expanding groupings of pixels to form a larger group and the uniformity and directionality of the operation can be controlled by the shape of the dilation mask. Likewise, erosion acts as the antithesis of dilation, shrinking groups of pixels to form tighter packages or possibly removing groups which are too small (noise and the like). Once again the exact effects of erosion will be determined by the choice of pixel mask used during the operation. By first dilating our image we are able to connect broken segments of our vertical laser stripe. Unfortunately this operation both increases the effects of noise left over from thresholding and makes our vertical stripe thicker than it was originally. Luckily eroding our image by slightly more than we dilated it (which is controlled through our choices of dilation and erosion masks) will allow us to remove both the harmful effects of the dilation while also removing noisy pixels all together and thinning our laser stripe to a narrower line.



After Dilate

At this point we reached another design choice: to use a topology preserving thinning operation much like those used in fingerprint recognition systems, or to use mean or median filtering to determine the center of our laser stripe. Mean filtering was highly susceptible to stray noise points far from the laser stripe in the image, and produced rather capricious results. Median filtering produced fairly accurate results, only failing when noise was present in vertical locations where the laser stripe was not present, and proved to be somewhat computationally efficient. Because of the success of median filtering, we determined that topology preserving thinning operations would be too high in computational cost to be worthwhile. Future works with this project may wish to explore thinning operations more thoroughly.



After Second Erode

A baseline image must also be considered each time the scanner (or any of its components are moved) to allow for a distance calculation between the axis of rotation and the laser stripe on the object. This distance calculation will be the basis for a coordinate system transformation which will be discussed in more detail in the following paragraphs. Our code can take two forms with regard to the baseline image: a more robust calculation which saves the baseline data and calculates a specific distance for each laser strip pixel with respect to the baseline, and a less robust form which assumes that the webcam is positioned with the vertical axis as a column within the image. The latter implementation is the one we use in practice because we have control of the webcam and feel that the added computational inefficiency of making the code robust to tilts in the camera angle is not worth our clock cycles.

After the images have been transformed into a binary edge mapping and the median horizontal line location is calculated, we can begin the coordinate system transformation calculations on the DSK. There are two coordinate systems to consider: the three dimensional coordinate system in which the object exists, and the two dimensional coordinate system of the image. By rotating the object's coordinate system through 360 degrees and knowing the angle of the laser pointer with respect to this rotating coordinate system, we provide ourselves with enough information to reconstruct the object in its own coordinate system. Below we have diagramed these two coordinate systems and the relationship between them:

Object/Turntable

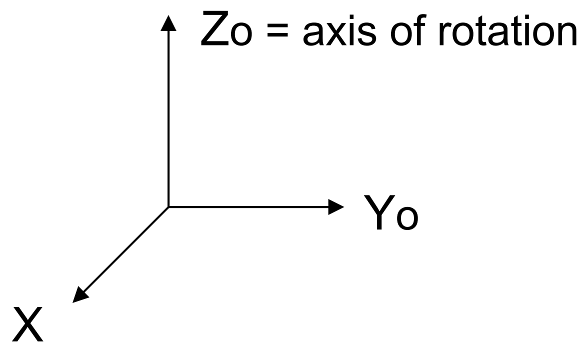
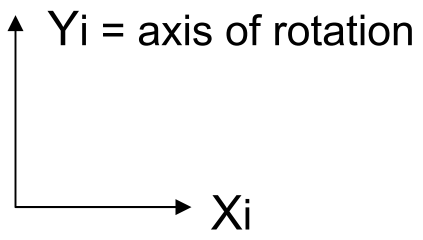
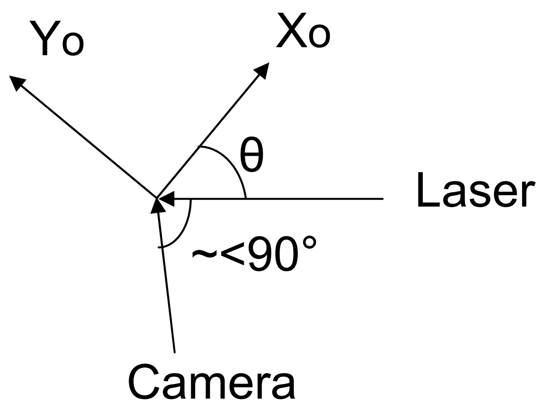


Image Coordinates



Top-Down View of Total System



From these diagrams we can determine the following relations:

Remember this is what we know:

θ, $X_i(\theta,\alpha)$, $Y_i(\theta, \alpha)$; (α is indexing variable)

And this is what we want:

$Z_o(\alpha)$, $Y_o(\alpha)$, $X_o(\alpha)$

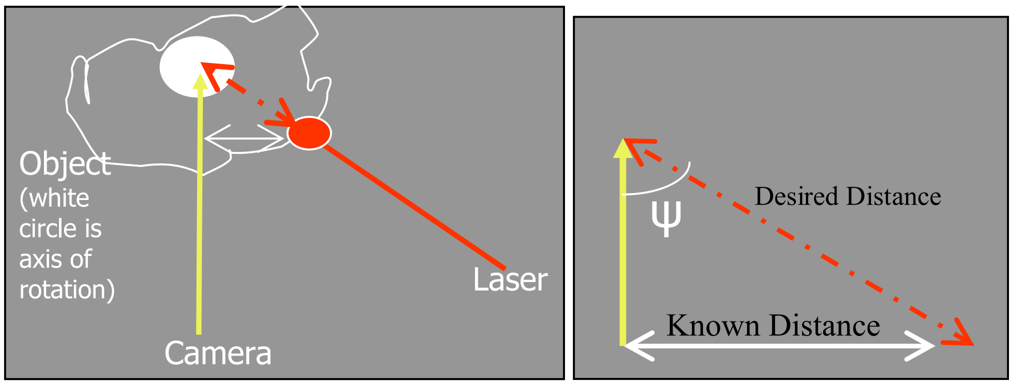
And this is the coordinate system transformation which will take the known data and compute what we want:

$$Z_o(\alpha) = Y_i(\theta, \alpha)$$

$$X_o(\alpha) = X_i(\theta,\alpha) \cos(\theta)$$

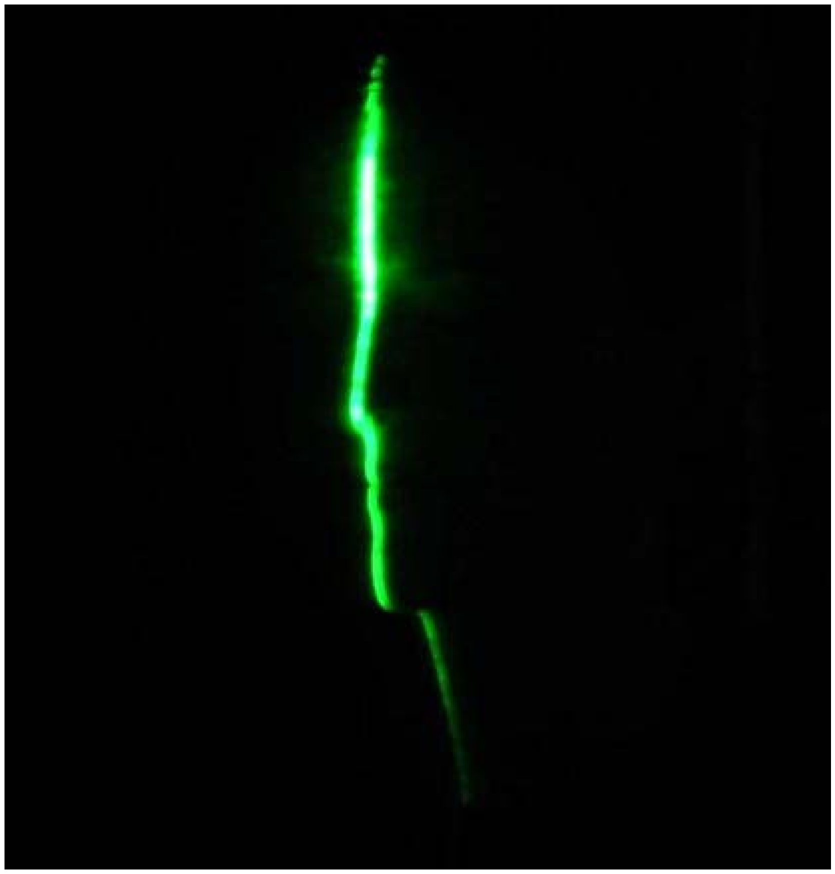
$$Y_o(\alpha) = Y_i(\theta, \alpha) \sin(\theta)$$

To detail the equations a little further, each $X_i(\theta,\alpha)$, $Y_i(\theta, \alpha)$ corresponds to a single point on the surface of the object, which is detailed by a binary edge point in our image. Using the trigonometric equations above, we use the $X_i(\theta,\alpha)$, $Y_i(\theta, \alpha)$, and the corresponding θ to transform the point back to the desired three dimensional coordinate system of the object. We also note that if a perfect profile is not achieved in our image (i.e. the webcam and the laser pointer are not at 90 degrees) and there is no compensation we will be effectively low-pass filtering our object in the horizontal direction. We derive the mathematics below to properly scale our image such that the object will have the proper proportions in our final output by the following trigonometric relation: the known distance divided by $\sin(\psi)$ is equal to the desired distance.



To visualize the image coordinate system we provide an example image below from a group who previously attempted a similar project who used a higher resolution webcam.

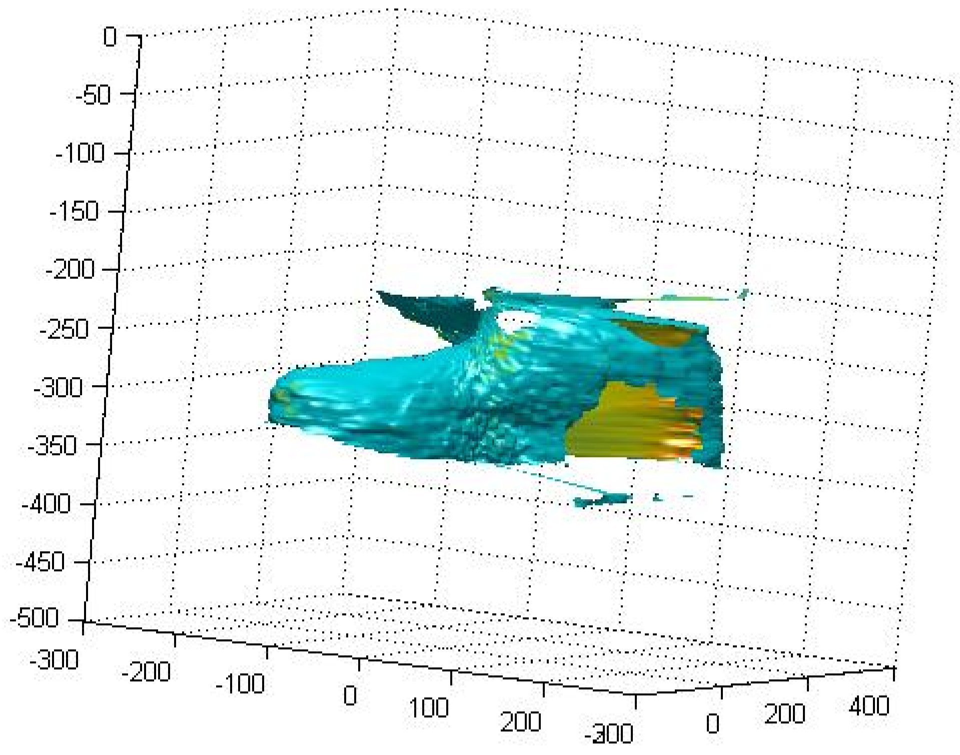
The image coordinate system before any processing is performed



(note the axis of rotation is to the right of the projected line, thus the X_i axis is $-X_i$ when the coordinate system is defined as above)

After finishing our coordinate system transformation we receive a fully rotational three dimensional plot, and a screen shot of an example output from our system is shown below:

Example of a Reconstructed Object



(Note the noisy points and segments of missing data, we will discuss why this shoe is one of the more difficult cases in more detail later)

Once the data is on the computer it is possible to convert the data to a file type which will allow 3 dimensional CAD programs to import the object for animation and rendering purposes.

Practical Programming Concerns

Real time concerns are not necessary in our project, but we still must consider the data pathways and storage limitations of our system. The data path originates in the webcam-

PC universal serial bus connection. This data path transfers 480 by 640 pixel images with 3 bytes per pixel every 1/15th of a second. This corresponds to a transfer rate of 13.8 Mbytes/sec. From this point the data is passed into our pre-processor stage where the data is stripped of two of the three color channels since the only color that is useful to the reconstruction is the laser color, in this case red. This is already a significant data reduction, but in order to further reduce the transfer size from PC to DSK, the data is thresholded. This threshold function reduces the data from a byte mapping to a binary mapping. Over all from camera to DSK, the PC pre-processing reduces the data size from 3 bytes per pixel down to 1 bit per pixel. To reiterate, this reduction in numbers is from 13.8 Mbytes/sec (straight from camera) to 4.6 Mbytes/sec (after channel stripping) to 4.6 Mbits/sec (after binary thresholding).

Once on the DSK, size requirements are a little more stringent, but the amount of data is still well within the allowable limits. Of the 16 Mbytes of SDRAM, the data for an entire scan fits compactly into 55.2 Mbits (for an average scan of 12 seconds) which corresponds to 6.9 Mbytes. This means that everything including the program easily fits onto the DSK. However due to the way we are utilizing our network we are transmitting one frame at a time from PC to DSK. The computation output size from the DSK back to PC is also not approaching the limits of the board. Each frame has 480 values which will then be transformed into 3 floats which are 4 bytes each. This comes out to 86.4 Kbytes/sec (480 pixels/frame * 15 frames/sec * 3 floats/pixel * 4 bytes/float) and since the average scan takes 12 seconds the entire output from the DSK will be 1036.8 Kbytes.

Interfaces and GUI

The graphical user interface we designed for this system is elegant and attractive, while maintaining a full range of functionality. The user is able to change key parameters within the algorithm to optimize the performance of the system for a variety of scanning conditions. These variables which may be tuned for a particular lighting environment include the thresholding value, and the color channel of the video to use (red, green, or blue depending on ambient lighting conditions and choice of laser beam wavelength). The other adjustable parameters which change the performance of the algorithm include a condition for the time spent on a rotation of an object, and a specification of the amount of rotation (in degrees) through which the object is scanned, allowing for robustness to a partial scan of an object like a face which may not have been scanned via the rotating platform. Functionally the graphical user interface includes a browser to specify the baseline image for the scanning system, and the video file of the scan itself. Additionally, a handy preview button and pane allow the user to see the selected baseline image with the amount of specified thresholding, allowing the user to more easily adjust the thresholding parameter for the lighting conditions present. When the program has completed and the data is transferred back from the DSK the graphical user interface will then launch a Matlab figure window displaying the three dimensional output of the scan. This Matlab figure has some key shading, lighting, and aspect ratios predetermined for easy viewing. The shading is interpolated to smooth the appearance of the object (which hides the usual mesh-like appearance), while the lighting is chosen to be overhead which will maintain consistency throughout rotation, and the aspect ratio is fixed such that the scale of the X, Y, and Z axis are one to one to one, thus preserving the natural shape of the scanned object. The figure is also set to automatically begin rotating

360 degrees to allow the user to preview the entire output of the scan, and once the 360 degree preview scan is completed, the user is able to freely zoom, rotate, illuminate, and move the object. These features of our interface are all enabled through the use of Java to create the graphical user interface. Because Matlab is a java built program, the interfacing required to produce the three dimensional final figure is straightforward. However, linking the interface with the C code and the DSK requires a java native interface (JNI). The java native interface allows us to easily program a GUI which will port the tuning parameters to the DSK C code while seamlessly connecting to Matlab with its native java programming language.

Conclusions and Improvements

We were successful in our efforts to build an extremely cheap laser scanner. The project does not require any unique or hard to find parts and produces fairly reliable results. While our output is in a Matlab friendly format, programs for porting Matlab figures to almost any other extension type are readily available on the internet. Upon a quick search we were able to import our models into both SolidWorks (a robust CAD structural simulator) and into 3D StudioMax (a common animation and modeling environment). These proof-of-concept imports show that once we have scanned and reconstructed the object, there are endless options available to the user.

As novice 3D StudioMax users we were able to develop a three dimensional scene with a scanned object sitting on a rendered table. We then created an animation of our scanned object sliding off the table and falling to the floor. As novice developers, there is no conceivable way we could model the object accurately in the CAD program. This example shows how our scanner is an ideal teaching tool in CAD programs, since beginner users could create more complicated and interesting scenery right away without any experience. Also the veteran CAD developers' would appreciate the speed at which a complicated object could be rendered to scale within the system. With an entire scan and three dimensional rendering taking less than two minutes, even the most experienced developer could not duplicate objects in the computer in the same amount of time. Thus, instead of spending time recreating objects in their systems, artists and animators can spend more time on developing fluid animations and complex scenery.

3D Studio Max Import of a Helmet

Artists and animators are not the only beneficiaries of our inexpensive and quick three dimensional scanner. After loading the scanned object into SolidWorks (as we were able to do) an engineer can easily perform mechanical analysis of the object. We were able to assign material properties to a scanned helmet, which would allow the engineer to perform preliminary analysis of the acceptable impact tolerances. This provides detailed analysis based upon the geometry of the structure making the integrity of the scan very important.

The limitations of our system are mostly the result of this desire for being inexpensive and could drastically be improved upon with a few purchases (i.e. a better camera or the use of two cameras). The level of detail that we can reconstruct is limited by the frame size and rate of the camera, as well as by the dispersion of the laser light. While the dispersion was not our limiting factor, a better camera would most certainly require a thinner laser strike to take advantage of the higher pixel count. The system itself requires adjustment through the GUI in every new lighting environment to reset the threshold levels as the ambient light changes.

A major source of error in our system is the loss of laser line of sight. Highly irregularly shaped objects cause the single camera to lose sight of the line. To obtain ideal data which requires no scaling factor, the laser line should be at a ninety degree angle to the camera. This situation provides the camera with a perfect profile of the scanning object, but irregularities in the shape cause the camera to lose sight of the laser line more often. By lowering the angle between the camera and the laser pointer, we effectively decreased the frequency of data loss at the cost of accuracies in the scaling factor (which theoretically fixes the problem perfectly, but because we are left estimating the actual angle between the camera and laser pointer, there is error introduced). The authors propose the introduction of a second camera into the system. By having the following system configuration, there will be no loss of data due to oblong objects and system accuracy will increase substantially on these objects.

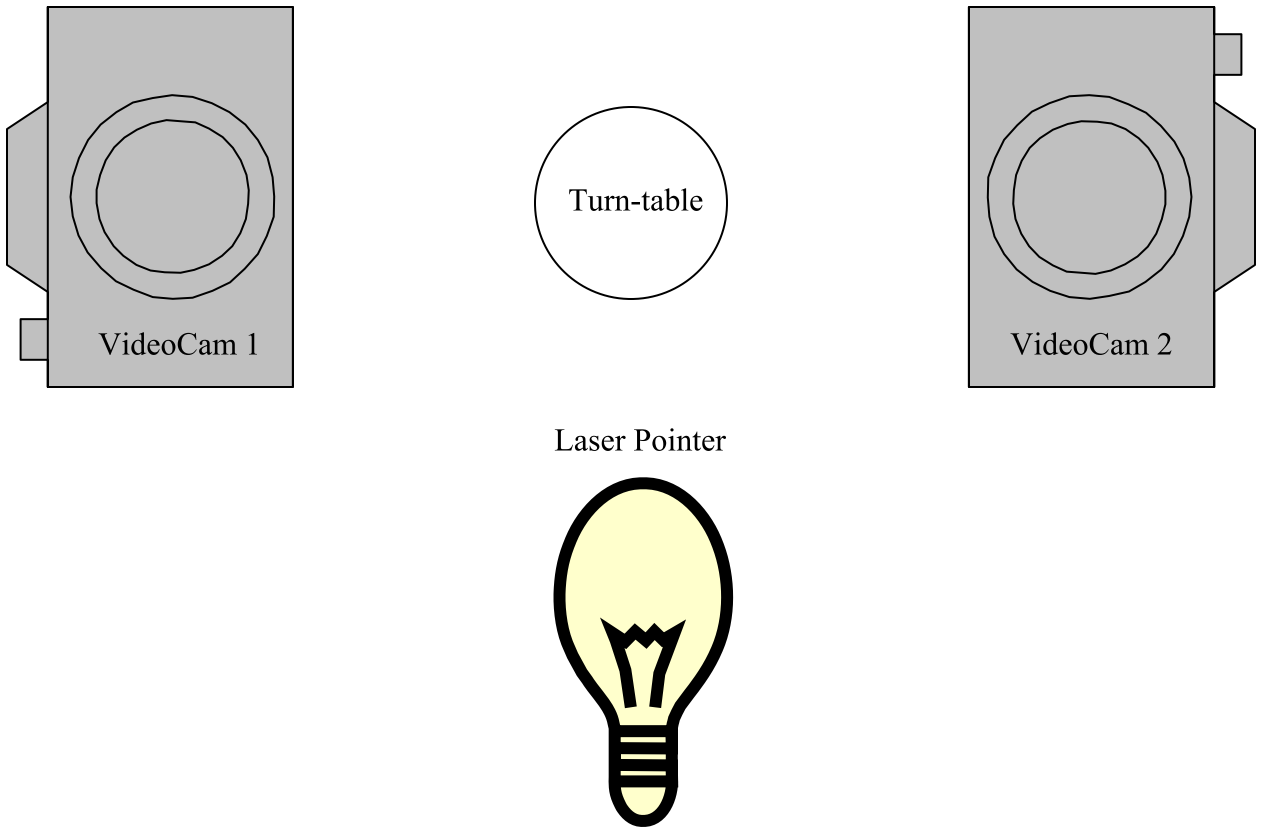


Figure 3. By including a second video camera, and keeping the laser pointer at a ninety degree angle to both, the system will lose no data due to line of sight of the laser line.

This system will provide more accurate results, however there are multiple new practical considerations. First, the cameras must have the exact same scale and tilt (which would actually be mirrored across the cameras) as each other, and the relative height of the object within the image must be considered. Secondly, the videos must be processed for the exact same times. For the latter problem the authors propose that the object is started rotating and the video is recording before the laser is turned on, and the proposed algorithm could then begin processing the videos at the first frame where the laser is switched on.

Another flaw in our system which could be addressed with future work is the inherent inability to model objects which are not continuous in form. For example, a coffee mug which has a handle. When the handle is in front of the camera's field of vision of the laser line, errors will be introduced into the system, and the current system is unable to model the gap between the mug and the outer portion of the handle (it would appear contiguous due to interpolation across neighboring data points). One possible proposed solution to this problem would be the use of lighting shadows or angular profiling. By placing a uniform backlight behind the object, the camera will receive a full profile image of the object, including gaps. Another advantage of this technique is the fact that a one-hundred and eighty degree rotation will provide the same amount of edge information as a single laser pointer would during a three-hundred and sixty degree rotation. The authors also note that combinations of the two previous suggestions provide many iterations of various solutions which could solve many of the problems in the system. For example, a system could use the configuration seen in the above figure, while adding another laser opposite the first, taking advantage of the fact that edge information from either side of the axis of rotation may be used, thus eliminating the need for three-hundred and sixty degree rotations.

Another inherent problem in our system is its inability to deal with objects with reflective surfaces or transparent materials. Several attempts were made to scan a metallic chalice with gems inlaid in its exterior. Though highly attractive to the eye, this object provided our system with much noise in the form of deflected laser light. Through clever image processing schemes it would be possible to consider noise reduction techniques which might be able to handle such problematic objects. The authors also attempted scans of a glass mug, whose video result was the most problematic. The glass surface disperses the laser throughout the object, and the glowing object becomes very difficult to profile accurately with our technique. The authors propose the use of filtering followed by edge detection techniques which might be able to extract the edge furthest from the axis of rotation for use in the reconstruction algorithm.

Due to the limitations of the camera's resolution, larger objects provide more accurate scans. Unfortunately our rotational platform can only handle a load of approximately two kilograms. In order to scan a human face, we require manual rotation techniques. Therefore the authors would expect vast improvements if a larger rotational platform were developed which could handle human sized objects. The effect of the laser line's halo will be ameliorated by the larger object, and deformations in the line will be more pronounced in a larger object, increasing the overall accuracy of the system. However, due to the geometry of the coordinate system transformations, more images per rotation are necessary to maintain the circumferential resolution of the system. There are two relatively easy solutions to this stumbling point: the object can be rotated slower, or the camera's frame rate can be increased. Because the authors' have been disappointed with the performance of the webcam, increasing the frame rate with the current rig is not an option, and slower rotation is the only answer. However, in future attempts with higher performance webcams, the authors suggest an increase in frame rate so the speed of acquisition is maintained.

Hardware/Purchases

All of our purchases, besides the paint, introduced problems into our project and we reverted to our previous setup. Firstly we bought the cylindrical acrylic dowel to replace our wine glass beam splitter, and then resorted back to the wine glass because of the random disconnects in the laser line due to the bubbles and internal debris. Then we bought the laser attachment to split the beam, and then resorted back to the wine glass because of the uniform disconnects in the laser line due to the beam splitters geometry. After conceiving grandiose rotating platform schemes with gearing and belts we resorted to a much simpler method of attaching the platform directly to a motor found in a garage. The webcam was generously provided by Professor Casasent. The red laser pointer was donated by a friend and the wine glass and wood were also lying around the house. Total useful purchases for this project came to be $3.00.

Purchases:

1. Laser Pointer Attachment Beam Splitter ~$12.00
2. Cylindrical Acrylic Dowel ~$2.00
3. Black Paint ~$3.00

Parts List:

1. DC Motor
2. Wine Glass
3. Red Laser Pointer
4. Web Cam
5. Random Pieces of Wood

<u>Total price</u>: ~$17.00

<u>Total price of parts used:</u> ~$3.00

Novelty

To our knowledge this project or closely related projects have never been attempted within 18-551. We are using the theoretical foundation of online hobbyists who have detailed their experiences with similar projects. (1 2 3) However, our project will be the first that we know of to utilize a DSP specific platform. We also used non-novel image processing techniques such as edge detection, but we derived the trigonometric linear algebra equations which will be used in the coordinate system transformations.

Database

There is neither a required database for this project nor any need to construct one. The data that we will be utilizing is obtained from real world models. If we are not able to construct our platform and obtain these required images, we have made contact with one of the hobbyists who performed a similar project and we hope they will be able to supply us with their raw pre-processed data.

Schedule

Oct 8-14	Acquire laser and camera (GJ & SN); Communications to DSK (GG); Rigging questions to machine shop (SN)
Oct 15-21	Motor and rotating platform parts acquired (SN); Beginning construction of rig (GJ&SN)
Oct 22-28	Get laser line and rotation working (GJ & SN)
Oct 29-Nov 4	Rig complete (GJ & SN); Coordinate system transformation (GJ&GG); Simple 6”x6”x6” object scanned (All);
Nov 5-11	Software communications (GG); Coordinate transforms (GJ); Edge detection (GG);
Nov 12-18	Working out kinks in rig (GJ & SN); Porting Matlab into C (GG)
Nov 19-25	Lots of coding & GUI (GG); Debugging code and rig (SN & GJ); Porting to 3D Studio Max (SN & GJ)
Nov 26- Dec 2	Begin practicing demo / writing final paper/ removing kinks (ALL)

References

Code Contribution:

1. **Argon. "3-D Scanner" available online:**
   http://www.instructables.com/id/ESQ41476M3EP285ZU2/?ALLSTEPS
2. **TI source code available online:**
   http://focus.ti.com/docs/toolsw/folders/print/sprc121.html
   http://focus.ti.com/docs/toolsw/folders/print/sprc094.html

System Implementation Details

3. **Steple. "3-D Laser Scanner" available online:**
   http://n.ethz.ch/student/pleiness/en/index.php
4. **Unkown. "3-D Scanning" available online:**
   http://www.thomasandamy.com/projects/Scan/

DSK Code

```
#include <csl.h>                /* chip support library initialization */
//#include <csl_cache.h>          /* cache library */
#include <csl_edma.h>           /* edma library */
#include <dsk6713.h>            /* dsk board library */
#include <stdio.h>              /* printf etc. */
#include <stdlib.h>             /* malloc etc. */
#include <dsknet.h>             /* DSK sockets library */
#include <img_dilate_bin.h>
#include <img_erode_bin.h>
#include <mathf.h>
/*-------------------------- Global variables -------------------------*/
/* Base address of DSK network daughter card */
#define SMC91C111_BASE    0xA0000000
unsigned int *SMC91C111 = (unsigned int *) SMC91C111_BASE;


char *bufIn;
char *bufOut;

/* EDMA */
static EDMA_Handle hEdma;
static EDMA_Config edmaConfig;
static int chaNum;
static int tcc;

/* TCP client socket */
SOCKET *socket;

/*---------------------------------------------------------------------*/

#define MIN(a,b) ((a < b) ? a:b)    /* Used in recvData() */
#define ABS(a) ((a>0.0) ? a: (-1*a))
#define inBufSz 38400
#define outBufSz 3840
#define PI 3.1416
char locBufIn[inBufSz];
char locBufIn2[inBufSz];
float locBufOut[outBufSz/4];
float locBufOut2[outBufSz/4];
int locMeds[480];
int base;
float adj;
int frames;
float angle;

/* Function prototypes */
void sendData(void *buf, int len);
void recvData(void *buf, int len);
void dmaCopyBlk(void *src, void *dst, Uint32 numBytes);

void printToFile(char* filename, unsigned char* ptr) {

}
void printToFile2(char* filename, unsigned char* ptr) {
        FILE* f;
        int i, j;
        unsigned char c;
        f = fopen(filename, "w");

        for(i=0; i<480; i++) {
                for(j=0; j<80; j++) {
                        c = ptr[i*80+j];
                        fprintf(f, "%d %d %d %d %d %d %d %d
", !!(c&0x01), !!(c&0x02), !!(c&0x04), !!(c&0x08), !!(c&0x10), !!(c&0x20), !!(c&0x40), !!(c&0x80));
                }
                fprintf(f, "\n");
        }
        fclose(f);
}

void moveL(char* ptr, int* eB, int*eb)
{
        int found = 0;
        while(!found)
```

```
		{
			(*eb)--;
			if((*eb) == -1)
			{
				(*eb) = 7;
				(*eB)--;
			}
			if((*eB) < 0)
				return;
			if(!ptr[(*eB)])
			{
				(*eB)--;
				(*eb)=8;
			}
			else
			{
				if(ptr[(*eB)] & 1<<(*eb))
					found = 1;
			}
		}
	}

	void moveR(char* ptr, int* sB, int*sb)
	{
		int found = 0;
		while(!found)
		{
			(*sb)++;
			if((*sb) == 8)
			{
				(*sb) = 0;
				(*sB)++;
			}
			if((*sB) > 80)
				break;
			if(!ptr[(*sB)])
			{
				(*sB)++;
```

```
                        (*sb)=-1;
                }
                else
                {
                        if(ptr[(*sB)] & 1<<(*sb))
                                found = 1;
                }
        }
}


// dilate mask which increases the pixel blobs in the vertical direction
// which will fill breaks in our laser line stripe
char dilateMsk1[] = {0, 1, 0, 0, 1, 0, 0, 1, 0};

// block erode mask which will remove noisy pixels left over from thresholding
char erodeMsk1[] = {1, 1, 1, 1, 1, 1, 1, 1, 1};

// vertical erode mask which will remove horizontally oriented pixels, effectively
// thinning our laser line
char erodeMsk2[] = {0, 1, 0, 0, 1, 0, 0, 1, 0};



void filter(void)
{
        int i, sB, sb, eB, eb, sCnt, eCnt;
        char* ptr;
        char* ptr2;
        ptr = locBufIn;
        ptr2 = locBufIn2+81;

        dmaCopyBlk(bufIn, locBufIn, inBufSz);
        while(!EDMA_intTest(tcc));

        printToFile("Zorig.dat", locBufIn);

        // erode away noisy pixels left over from thresholding
        for(i=1; i<480-1; i++) {
            IMG_erode_bin(ptr, ptr2, erodeMsk1, 80);
            ptr+=80;
            ptr2+=80;
}

printToFile("Zerode1.dat", locBufIn2);

ptr = locBufIn2;
ptr2 = locBufIn+81;

memset(locBufIn, 0, inBufSz);

// dilate to fill gaps in laser line
for(i=1; i<480-1; i++) {
            IMG_dilate_bin(ptr, ptr2, dilateMsk1, 80);
            ptr+=80;
            ptr2+=80;
}

printToFile("Zdilate.dat", locBufIn);

ptr = locBufIn;
ptr2 = locBufIn2+81;

// erode away horizontal pixels to thin laser line
for(i=1; i<480-1; i++) {
            IMG_erode_bin(ptr, ptr2, erodeMsk2, 80);
            ptr+=80;
            ptr2+=80;
}

printToFile("Zerode2.dat", locBufIn2);



ptr = locBufIn2;

for(i=0; i<480; i++) {
            sB = 0;
```

```
				sb = -1;
				eB = 79;
				eb = 8;
				sCnt=eCnt=0;
				while(eB > 0 && !(eB<=sB && eb<=sb)) {
					if(eCnt <= sCnt) {
						moveL(ptr,&eB,&eb);
						eCnt++;
					}
					else {
						moveR(ptr, &sB, &sb);
						sCnt++;
					}
				}
				if(eB != sB || eb != sb) {
					locMeds[i] = 0;
				}
				else {
					locMeds[i] = sb+8*sB;
				}
				ptr+=80;
			}
		}

		float fil[] = {0.0625, 0.125, 0.1875, 0.25, 0.1875, 0.125, 0.0625};

		// translate
		void translate(int i)
		{
			int j,k;
			float* ptr;
			float c, s, t;
			ptr = locBufOut;
			t = ((float)i);
			t = angle*t / ((float)frames);
			c = adj*cosf(t);
			s = adj*sinf(t);
			//printf("c: %f, s: %f, t: %f\n", c, s, t);
```

```
	for(j=0; j<480; j++) {
		if(locMeds[j] == 0) {
			ptr[j] = 0;
			ptr[j+480] = 0;
		}
		else {
			k = base-locMeds[j];
//			printf("  K: %d", k);
//			printf("  ABS(K): %d\n", k);
			if(k > 0) {
				t = (float)k;
				ptr[j] = c*t;
				ptr[j+480] = s*t;
			}
		}
	}


	locBufOut2[0] =
(fil[0]+fil[1]+fil[2]+fil[3])*locBufOut[0]+fil[4]*locBufOut[1]+fil[5]*locBufOut[2]+fil[6]*locBufOut[3];
	locBufOut2[1] = (fil[0]+fil[1]+fil[2])*locBufOut[0] +
fil[3]*locBufOut[1]+fil[4]*locBufOut[2]+fil[5]*locBufOut[3]+fil[6]*locBufOut[4];
	locBufOut2[2] =
(fil[0]+fil[1])*locBufOut[0]+fil[2]*locBufOut[1]+fil[3]*locBufOut[2]+fil[4]*locBufOut[3]+fil[5]*locBuf
Out[4]+fil[6]*locBufOut[5];
	locBufOut2[480] =
(fil[0]+fil[1]+fil[2]+fil[3])*locBufOut[480]+fil[4]*locBufOut[481]+fil[5]*locBufOut[482]+fil[6]*locBuf
Out[483];
	locBufOut2[481] = (fil[0]+fil[1]+fil[2])*locBufOut[480] +
fil[3]*locBufOut[481]+fil[4]*locBufOut[482]+fil[5]*locBufOut[483]+fil[6]*locBufOut[484];
	locBufOut2[482] =
(fil[0]+fil[1])*locBufOut[480]+fil[2]*locBufOut[481]+fil[3]*locBufOut[482]+fil[4]*locBufOut[483]+fil[
5]*locBufOut[484]+fil[6]*locBufOut[485];
	for(j=3; j<477; j++) {
		locBufOut2[j] = fil[0]*locBufOut[j-3]+fil[1]*locBufOut[j-2]+fil[2]*locBufOut[j-
1]+fil[3]*locBufOut[j]+fil[4]*locBufOut[j+1]+fil[5]*locBufOut[j+2]+fil[6]*locBufOut[j+3];
```

```
                        locBufOut2[480+j] =
fil[0]*locBufOut[477+j]+fil[1]*locBufOut[478+j]+fil[2]*locBufOut[479+j]+fil[3]*locBufOut[480+j]+fil
[4]*locBufOut[481+j]+fil[5]*locBufOut[482+j]+fil[6]*locBufOut[483+j];
            }
            locBufOut2[477] =
fil[0]*locBufOut[474]+fil[1]*locBufOut[475]+fil[2]*locBufOut[476]+fil[3]*locBufOut[477]+fil[4]*locB
ufOut[478]+(fil[5]+fil[6])*locBufOut[479];
            locBufOut2[478] =
fil[0]*locBufOut[475]+fil[1]*locBufOut[476]+fil[2]*locBufOut[477]+fil[3]*locBufOut[478]+(fil[4]+fil[5
]+fil[6])*locBufOut[479];
            locBufOut2[479] =
fil[0]*locBufOut[476]+fil[1]*locBufOut[477]+fil[2]*locBufOut[478]+(fil[3]+fil[4]+fil[5]+fil[6])*locBuf
Out[479];
            locBufOut2[957] =
fil[0]*locBufOut[954]+fil[1]*locBufOut[955]+fil[2]*locBufOut[956]+fil[3]*locBufOut[957]+fil[4]*locB
ufOut[958]+(fil[5]+fil[6])*locBufOut[959];
            locBufOut2[958] =
fil[0]*locBufOut[955]+fil[1]*locBufOut[956]+fil[2]*locBufOut[957]+fil[3]*locBufOut[958]+(fil[4]+fil[5
]+fil[6])*locBufOut[959];
            locBufOut2[959] =
fil[0]*locBufOut[956]+fil[1]*locBufOut[957]+fil[2]*locBufOut[958]+(fil[3]+fil[4]+fil[5]+fil[6])*locBuf
Out[959];

            dmaCopyBlk(locBufOut2, bufOut, outBufSz);
            while(!EDMA_intTest(tcc));
}

/*************************************************************************
 * Name: main
 ************************************************************************/
int main(void)
{
            int i, tot, cnt, b, a;
    /* Initialize the DSK network daughter card and its network library */
    if ( !dsk91c111_init(AUTO_NEG) )
    {
        printf("Error in dsk91c111_init().  Exiting...\n");
        printf("Reload the program or restart CCS.\n");
        exit(1);
    }
    if ( !net_init("DSK", DHCP_ENABLE, NULL, NULL) )
    {
        printf("Error in net_init().  Exiting...\n");
        printf("Not enough free heap memory.\n");
        exit(2);
    }

    while(1) {
            socket = socket_open(ANY_ADDRESS, ANY_PORT, 5031, DATATYPE_CHAR,
TCP_INIT_FUNC);
            if(socket == NULL) {
                    printf("Error in socket_open(). Exiting...\n");
                    printf("Not enough free heap memory.\n");
                    exit(3);
            }

    /* Initialize the chip support library, must when using CSL */
    CSL_init();
    /* Open an EDMA channel */
    hEdma = EDMA_open(EDMA_CHA_ANY, EDMA_OPEN_RESET);
    chaNum = (hEdma & 0x00FF0000) >> 16;
    /* Allocate an EDMA transfer-complete code (TCC) */
    tcc = EDMA_intAlloc(-1);

    /* Enable caching of SDRAM memory */
//    CACHE_enableCaching(CACHE_CE00);
    /* Use 32kB of L2 cache */
//    CACHE_setL2Mode(CACHE_32KCACHE);

    /* Initialize the board support library, must be called first */
    DSK6713_init();

            bufIn = malloc(inBufSz);
            if(bufIn == NULL) {
                    printf("bufIn failed to malloc....exiting\n");
                    return 3;
            }
            bufOut = malloc(outBufSz);
            if(bufOut == NULL) {
                    printf("bufOut failed to malloc....exiting\n");
                    return 3;
            }
    if( !accept(socket) ) {
                    printf("Rdy\n");
                    while( !accept(socket)) {
                            net_recv(socket, bufIn, 0);
                            net_isq();
                    }
            }

            memset(locMeds, 0, 480*sizeof(int));
            memset(locBufIn, 0, inBufSz);
            memset(locBufIn2, 0, inBufSz);
            memset(locBufOut, 0, outBufSz);

            printf("Running\n");

            recvData(bufIn, 16);
            frames = ((int*)bufIn)[0];
            angle = ((float*)bufIn)[1];
            b = ((int*)bufIn)[2];
            a = ((float*)bufIn)[3];

            adj = 1.0/sinf(PI/4.0);

            sendData(bufOut, 4);

            recvData(bufIn, inBufSz);
            filter();
            tot = 0;
            cnt = 0;
            for(i=0; i<480; i++) {
                    if(locMeds[i]!=0) {
                            tot+=locMeds[i];
                        cnt++;
                    }
                }
                base = tot/cnt;
                if(b>0)
                    base = b;
//              printf("%d, %d\n",cnt, tot);
                sendData(bufOut, 1*sizeof(int));

                for(i = 0; i < frames; i++) {
                    memset(locMeds, 0, 480*sizeof(int));
                    memset(locBufIn, 0, inBufSz);
                    memset(locBufIn2, 0, inBufSz);
                    memset(locBufOut, 0, outBufSz);

                    recvData(bufIn, inBufSz);
                    filter();
                    translate(i);
                    sendData(bufOut, outBufSz);
                }

                printf("Finished\n");

                free(bufIn);
                free(bufOut);
    /* Free the EDMA transfer-complete code (TCC) */
    EDMA_intFree(tcc);
    /* Close the EDMA channel */
    EDMA_close(hEdma);
    /* Shut down and close the socket */
    while ( !shutdown(socket, 10000) )
    {
        printf("Error in shutdown(): %s",
            error_tab[socket->error_code]);
    }
    if ( !socket_close(socket) )
    {
        printf("Socket has already been closed.\n");
    }
    printf("Exiting program...\n");


    }
            return 0;
}

/*------------------------ FUNCTION DEFINITIONS -----------------------*/

/************************************************************************
 * Name:    sendData
 * Purpose: Sends data to the PC.  'buf' is the pointer to the location of
 *   of the data on the DSK, 'len' is the length of the data.
 ************************************************************************/
void sendData(void *buf, int len)
{
    if (net_send_ready(socket, buf, len/sizeof(char), 1000000) == 0)
    {
        printf("Error in sendData().\n");
        printf("Error in net_send_ready(): %s\n",
            error_tab[socket->error_code]);
    }
}

/************************************************************************
 * Name:    recvData
 * Purpose: Receives data from the PC.  'buf' is the pointer to the buffer
 *   on the DSK in which to write the incoming data, 'len' is the length of
 *   the incoming data.
 ************************************************************************/
void recvData(void *buf, int len)
{
    char *ptr;
    int32_t amtToXfer;
    int32_t amtXferred;

    len = len / sizeof(char);
    ptr = buf;
    amtXferred = 0;
    amtToXfer = len - amtXferred;
    amtToXfer = MIN(1400, amtToXfer);

    while (amtXferred < len)
    {
        if (net_recv_ready(socket, ptr, &amtToXfer, 1000000) < 0)
        {
            printf("Error in recvData().\n");
            printf("Error in net_recv_ready(): %s\n",
                error_tab[socket->error_code]);
        }
        else
        {
            ptr += amtToXfer;
            amtXferred += amtToXfer;
            amtToXfer = len - amtXferred;
            amtToXfer = MIN(1400, amtToXfer);
        }
    }

    /* Send acknowledgment message to the PC */
    net_send_ready(socket, &amtXferred, 4/sizeof(char), 1000000);
}

/**************************************************************************
 * Name:    dmaCopyBlk
 * Purpose: Copies 'numBytes' from 'src' to 'dst'.  This function is
 *   ASYNCHRONOUS!  You must poll EDMA_intTest(tcc) to see when the
 *   transfer is complete.
 *   'numBytes' must be a multiple of 4 and must be <= 4 * 0xFFFF.
 **************************************************************************/
void dmaCopyBlk(void *src, void *dst, Uint32 numBytes)
{
    /* Frame synchronized 1D-to-1D transfers
     * src and dst addresses are incremented as needed
     * transfer completion interrupt is enabled using
     *   tcc as the transfer-complete code */
    edmaConfig.opt = 0x21300001 | (tcc << 16);

    edmaConfig.src = (Uint32) src;
    edmaConfig.cnt = numBytes/4;     /* One frame and 4-byte elements */
    edmaConfig.dst = (Uint32) dst;
    edmaConfig.idx = 0; /* Don't care */
    edmaConfig.rld = 0; /* Don't care */

    /* Set up the EDMA channel using the above configuration structure */
    EDMA_config(hEdma, &edmaConfig);

    /* Clear the EDMA transfer-complete flag */
    EDMA_intClear(tcc);

    /* Submit the EDMA transfer request */
    EDMA_setChannel(hEdma);
}
```

PC Code

```
#include "DSKObject.h"
#define PI 3.1416

int preview(int channel, const char* vidFile, int threshold, int base)
{
      AVIObject* video = new AVIObject(vidFile);
      unsigned char* buf = (unsigned char*)video-
>getThresholdedFrame(base, threshold, channel);
      FILE* f = fopen("prev.dat", "w");
      for(int i=0; i<38400; i++) {
            fprintf(f, "%d\n", (int)buf[i]);
      }
      fclose(f);
      return 0;
}

int run(int channel, int threshold, const char* vidFile, int angle, int
startFrame, float duration, int base, int b, int a) {
      DSKObject* com = new DSKObject();
      return com->run2(channel, threshold, vidFile, angle, startFrame,
duration, base, b, a);
}

DSKObject::DSKObject()
{
    WSADATA wsaData;
    WORD wVersionRequested;
    int err;
    SOCKADDR_IN serverInfo;

    /* Initialize Winsock library */
    wVersionRequested = MAKEWORD(2,2);
    err = WSAStartup(wVersionRequested, &wsaData);
    if(err) return;

    /* Create a TCP socket to listen for an incoming connection from
the
     * DSK client */
    connectSocket = socket(AF_INET,      /* Use IPv4 */
                          SOCK_STREAM,  /* Connection-oriented socket
*/
                          IPPROTO_TCP); /* Use TCP */

    if (connectSocket == INVALID_SOCKET)
    {
        WSACleanup();
            return;
    }

    /* Specify the the address family, IP address, and port number of
the
     * DSK server to which the PC client should connect */
    serverInfo.sin_family = AF_INET;    /* Use IPv4 */
    /* DSK IP address */
    serverInfo.sin_addr.s_addr = inet_addr("192.168.0.101");
    /* DSK server port number (it is set up in the DSK-side code) */
    serverInfo.sin_port = htons(5031);
    /* Connect to the DSK server */

    if (connect(connectSocket, (SOCKADDR *) &serverInfo,
        sizeof(serverInfo)) == SOCKET_ERROR)
    {
        closesocket(connectSocket);
        WSACleanup();
    }
}
DSKObject::~DSKObject() {
    closesocket(connectSocket);
    WSACleanup();
}


int DSKObject::run2(int channel, int threshold, const char* vidFile,
int angle, int startFrame, float duration, int base, int b, int a)
{
      float* res = (float*)malloc(3840);


      unsigned char* buf;
      AVIObject* video = new AVIObject(vidFile);
      buf = (unsigned char*)malloc(16);

      int frames = (int)(15.0*duration);

      ((int*)buf)[0] = frames;
      ((int*)buf)[2] = b;
      ((float*)buf)[1] = (float)(angle*PI/180.0);
      ((float*)buf)[3] = (float)(a*PI/180.0);
      sendData(buf, 16);
      recvData(buf, 4);
      free(buf);

      buf = (unsigned char*)video->getThresholdedFrame(base, threshold,
channel);

      sendData(buf, 38400);
      recvData(res, 4);

      FILE* X = fopen("X.txt", "w");
      FILE* Y = fopen("Y.txt", "w");
      int j;
      int k = startFrame+frames;
      for(int i=startFrame; i<k; i++)
      {
            // Threshold video data on PC side
            buf = (unsigned char*)video->getThresholdedFrame(i,
threshold, channel);

            // send thresholded image to the DSK
            sendData(buf, 38400);

            // recieve 3D data
            recvData(res, 3840);

            // create a X and Y matrix with the rows corresponding to
the frame of the video sequence
```

```
            // and the columns corresponding to the Z axis location of
the point
            for(j=0; j<480; j++) {
                  fprintf(X, "%f ", res[j]);
            }
            fprintf(X, "\n");
            for( ; j<960; j++) {
                  fprintf(Y, "%f ", res[j]);
            }
            fprintf(Y, "\n");

      }

      fclose(X);
      fclose(Y);

      /* Cleanup code */
    free(res);
    printf("Closing socket and exiting...\n");
    return 0;
}

/*------------------------ FUNCTION DEFINITIONS ---------------------
--*/


/******************************************************************
****
 * Name:    sendData
 * Purpose: Sends data to the DSK.  'buf' is the pointer to the
location of
 *   of the data on the PC, 'len' is the length of the data.

*******************************************************************
**/
void DSKObject::sendData(void *buf, int len)
{
    char *ptr;
    int ret;
    int amtXferred;

    ptr = (char*)buf;
    amtXferred = 0;

    while (amtXferred < len)
    {
        ret = send(connectSocket, ptr, len-amtXferred, 0);

        if (ret == SOCKET_ERROR)
        {
            printf("Error in sendData().\n");
            printf("Error in send(): %d.  Exiting...\n",
                WSAGetLastError());

            closesocket(connectSocket);
            WSACleanup();
            exit(1);
        }
          else
        {
            amtXferred += ret;
            ptr += ret;
        }
    }

    /* Receive acknowledgment message from the DSK */
    recv(connectSocket, (char *) &amtXferred, 4, 0);
}

/***********************************************************************
****
 * Name:    recvData
 * Purpose: Receives data from the DSK.  'buf' is the pointer to the
buffer
 *   on the PC in which to write the incoming data, 'len' is the length
of
 *   the incoming data.

***********************************************************************
**/
void DSKObject::recvData(void *buf, int len)
{
    char *ptr;
    int ret;
    int amtXferred;

    ptr = (char*)buf;
    amtXferred = 0;

    while (amtXferred < len)
    {
        ret = recv(connectSocket, ptr, len-amtXferred, 0);

        if (ret == SOCKET_ERROR)
        {
            printf("Error in recvData().\n");
            printf("Error in recv(): %d.  Exiting...\n",
                WSAGetLastError());

            closesocket(connectSocket);
            WSACleanup();
            exit(1);
        }
          else
        {
            amtXferred += ret;
            ptr += ret;
        }
    }
}
```

```
#include ".\aviobject.h"

AVIObject::AVIObject(LPCTSTR szFile)
{
        HRESULT result;
        AVIFileInit();
        result = AVIFileOpen(&pfile, szFile, OF_READ, NULL);
        if(result != 0) {
                printf("Error:\n");
                printf("AVIObject::AVIOjbec(LPCTSTR szFile)\n");
                printf("result = AVIFileOpen(&pfile, szFile, OF_READ, NULLL);\n");
                printf("errorno = %d\n", result);
        }
        result = AVIFileInfo(pfile, &fi, sizeof(fi));
        if(result != 0) {
                printf("Error:\n");
                printf("AVIObject::AVIOjbec(LPCTSTR szFile)\n");
                printf("result = AVIFileInfo(pfile, &fi, sizeof(fi));\n");
                printf("errorno = %d\n", result);
        }
        result = AVIFileGetStream(pfile, &pavi, streamtypeVIDEO, 0);
        if(result != 0) {
                printf("Error:\n");
                printf("AVIObject::AVIOjbec(LPCTSTR szFile)\n");
                printf("result = AVIFileGetStream(pfile, &pavi, streamtypeVIDEO, 0);\n");
                printf("errorno = %d\n", result);
        }
        result = AVIStreamInfo(pavi, &psi, sizeof(psi));
        if(result != 0) {
                printf("Error:\n");
                printf("AVIObject::AVIOjbec(LPCTSTR szFile)\n");
                printf("result = AVIStreamInfo(pavi, &psi, sizeof(psi));\n");
                printf("errorno = %d\n", result);
        }
        pgf = AVIStreamGetFrameOpen(pavi, NULL);
        if(pgf == NULL) {
                printf("Error:\n");
                printf("AVIObject::AVIOjbec(LPCTSTR szFile)\n");
```

```
		printf("pgf = AVIStreamGetFrameOpen(pavi, NULL);\n");
		printf("pgf = NULL - this means we were unable to find a codec\n");
	}
	thOutput = (unsigned char*)malloc(480*80);
	memset(thOutput, 0, 480*80);
}

AVIObject::~AVIObject(void)
{
	free(thOutput);
	AVIStreamGetFrameClose(pgf);
	AVIStreamRelease(pavi);
	AVIFileRelease(pfile);
	AVIFileExit();
}

void* AVIObject::getFrame(int i)
{
	LPBITMAPINFOHEADER lpbi;
	lpbi = (LPBITMAPINFOHEADER)AVIStreamGetFrame(pgf, i);
	return (lpbi->biSize)+(char*)lpbi;
}

void* AVIObject::getThresholdedFrame(int i, int th, int off)
{
	int len = 640*480*3;
	unsigned short fil = 1<<15;
	unsigned char* buf = (unsigned char*)getFrame(i);
	unsigned char temp;
	int j,k;
	for(i=0, j=0; i<len; i+=24, j++)
	{
		temp = 0;
		if(buf[off+i+21] > th) temp |= 0x80;
		if(buf[off+i+18] > th) temp |= 0x40;
		if(buf[off+i+15] > th) temp |= 0x20;
		if(buf[off+i+12] > th) temp |= 0x10;
		if(buf[off+i+9 ] > th) temp |= 0x08;
```

```
                if(buf[off+i+6 ] > th) temp |= 0x04;
                if(buf[off+i+3 ] > th) temp |= 0x02;
                if(buf[off+i   ] > th) temp |= 0x01;
                thOutput[j] = temp;
        }
        return thOutput;
}
```

GUI CODE

```
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GraphicsEnvironment;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.Color;
import java.awt.Graphics;
import java.io.BufferedReader;
import java.io.FileReader;

import java.net.URL;
import java.io.File;

import javax.media.*;

import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

public class SAP extends JFrame implements ActionListener {
	private static final long serialVersionUID = 1L;
	private ImagePanel previewImg;
	static {
		System.loadLibrary("SAP");
	}
	native int run(int channel, int threshold, String vidFile, int angle, int startFrame, float duration,
int base, int b, int a);
	native int preview(int channel, String vidFile, int threshold, int base);

	private JButton preview, go;
	private JFileChooser test;
	private JLabel channelL, thresholdL, videoFileL, baseFileL, angleL, startFrameL, durationL,
b2L, aL;
	private JTextField channelTF, thresholdTF, videoFileTF, baseFileTF, angleTF, startFrameTF,
durationTF, b2TF, aTF;

	private String defaultText = "defaultText";

	public static void main(String[] argv) {
		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				new SAP();
			}
		});
	}

	public void actionPerformed(ActionEvent arg0) {
		Object source = arg0.getSource();
		if (source == go)
		{
			try
			{
//				PlayMP3Thread player = new PlayMP3Thread(new
File("mp3.mp3"));
				int channel = Integer.parseInt(channelTF.getText());
				int threshold = Integer.parseInt(thresholdTF.getText());
				String vidFile = videoFileTF.getText();
				int base = 15*(int)Double.parseDouble(baseFileTF.getText());
				int angle = Integer.parseInt(angleTF.getText());
				int startFrame =
15*(int)Double.parseDouble(startFrameTF.getText());
				float duration = (float)Double.parseDouble(durationTF.getText());
				int b = Integer.parseInt(b2TF.getText());
```

```
					int a = Integer.parseInt(aTF.getText());
					int result = run(channel, threshold, vidFile, angle, startFrame,
duration, base, b, a);
//					player.stopPlay();
				}
				catch(Exception e)
				{
					System.out.println(e);
				}
				System.exit(0);
			}
			else if(source == preview)
			{
				try
				{
					int channel = Integer.parseInt(channelTF.getText());
					int threshold = Integer.parseInt(thresholdTF.getText());
					String vidFile = videoFileTF.getText();
					int base = 15*(int)Double.parseDouble(baseFileTF.getText());
					int result = preview(channel, vidFile, threshold, base);
					previewImg.read();
				}
				catch(Exception e)
				{
					System.out.println(e);
				}
			}
		}

		public SAP() {
			super("Strike A Pose");

			go = new JButton("GO");
			go.setFont(new Font("Stencil", Font.PLAIN, 250));
			preview = new JButton("Preview");
			preview.addActionListener(this);
			go.addActionListener(this);
```

```
channelTF = new JTextField("0");
thresholdTF = new JTextField("80");
videoFileTF = new JTextField("c:/data/Video1.avi");
baseFileTF = new JTextField("1.0");
angleTF = new JTextField("360");
startFrameTF = new JTextField("5.0");
durationTF = new JTextField("10.8");
b2TF = new JTextField("0");
aTF = new JTextField("45");

Font fontLabels = new Font("Script MT Bold", Font.PLAIN, 45);
channelL = new JLabel("Channel");
channelL.setFont(fontLabels);
thresholdL = new JLabel("Threshold");
thresholdL.setFont(fontLabels);
videoFileL = new JLabel("Video File");
videoFileL.setFont(fontLabels);
baseFileL = new JLabel("Base Image (s)");
baseFileL.setFont(fontLabels);
angleL = new JLabel("Angle");
angleL.setFont(fontLabels);
startFrameL = new JLabel("Start Frame (s)");
startFrameL.setFont(fontLabels);
durationL = new JLabel("Duration (s)");
durationL.setFont(fontLabels);
b2L = new JLabel("Base Image Default");
b2L.setFont(fontLabels);
aL = new JLabel("Laser/Cam Angle");
aL.setFont(fontLabels);

Container main = getContentPane();
main.setLayout(new BoxLayout(main, BoxLayout.Y_AXIS));
//MAIN BOX
Box mainBox = Box.createHorizontalBox();
main.add(mainBox);
//LABEL BOX
Box labelBox = Box.createVerticalBox();
labelBox.add(channelL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(thresholdL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(videoFileL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(baseFileL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(angleL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(startFrameL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(durationL);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(b2L);
labelBox.add(Box.createVerticalStrut(10));
labelBox.add(aL);
labelBox.add(Box.createVerticalStrut(10));
mainBox.add(labelBox);

channelTF = new JTextField("0");
thresholdTF = new JTextField("80");
videoFileTF = new JTextField("c:/Video1.avi");
baseFileTF = new JTextField("1.0");
angleTF = new JTextField("360");
startFrameTF = new JTextField("5.0");
durationTF = new JTextField("10.8");
//TEXTFIELD BOX
Box textFieldBox = Box.createVerticalBox();
textFieldBox.add(channelTF);
textFieldBox.add(thresholdTF);
textFieldBox.add(videoFileTF);
textFieldBox.add(baseFileTF);
textFieldBox.add(angleTF);
textFieldBox.add(startFrameTF);
textFieldBox.add(durationTF);
textFieldBox.add(b2TF);
textFieldBox.add(aTF);
mainBox.add(textFieldBox);
```

```
                //PREVIEW BOX
                Box previewBox = Box.createVerticalBox();
                previewImg = new ImagePanel();
                previewBox.add(previewImg);
                previewBox.add(preview);
                mainBox.add(previewBox);
                //GO
                main.add(go);


/*              leftBox.add(comp)
                box = Box.createHorizontalBox();
                box.add(Box.createHorizontalBox());
                main.add(box);
                box.add(Box.createHorizontalStrut(90));
                box.add(connectButton);
                box.add(Box.createHorizontalStrut(30));
                box.add(disconnectButton);
                box.add(Box.createHorizontalStrut(90));

                main.add(Box.createVerticalStrut(30));

                box = Box.createHorizontalBox();
                main.add(box);
                box.add(Box.createHorizontalStrut(30));
                box.add(certaintyPlanButton);
                box.add(Box.createHorizontalStrut(30));
                box.add(uncertaintyPlanButton);
                box.add(Box.createHorizontalStrut(30));

                box = Box.createHorizontalBox();
                main.add(box);
                main.add(Box.createVerticalStrut(30));

                box.add(textField1);

                box = Box.createHorizontalBox();
                main.add(box);
                main.add(Box.createVerticalStrut(30));

                box.add(textField2);

                main.add(Box.createVerticalStrut(30));

                box = Box.createHorizontalBox();
                main.add(box);
                box.add(stopButton);
                box.add(Box.createHorizontalStrut(30));
                box.add(quitButton);

                main.add(Box.createVerticalStrut(30));
*/

                repaint();

                this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
                this.addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                                // quit();
                        }
                });

                this.setLocationByPlatform(true);
                this.pack();
                this.setVisible(true);

        }

        class ImagePanel extends JPanel {

                public ImagePanel() {
              setPreferredSize(new Dimension(640, 480));
                  setBackground(Color.black);
                  read = false;
                }
```

```
                    boolean read;

                @Override public void paintComponent(Graphics g) {
                  super.paintComponent(g);
                        g.drawLine(200,200, 1, 1);
                        if(read) {
                        try {
                              g.setColor(Color.white);
                              BufferedReader br = new BufferedReader(new
FileReader("prev.dat"));

                              for(int j=480; j>=0; j--) {
                                    for(int i=0; i<640; i+=8) {
                                          int tot = Integer.parseInt(br.readLine());
                                          if(tot >= 128) {
                                                g.drawLine(i+7,j, i+7, j);
                                                tot -= 128;
                                          }
                                          else {
                                                g.setColor(Color.black);
                                                g.drawLine(i+7, j, i+7, j);
                                                g.setColor(Color.white);
                                          }
                                          if(tot >= 64) {
                                                g.drawLine(i+6,j, i+6, j);
                                                tot -= 64;
                                          }
                                          else {
                                                g.setColor(Color.black);
                                                g.drawLine(i+6, j, i+6, j);
                                                g.setColor(Color.white);
                                          }
                                          if(tot >= 32) {
                                                g.drawLine(i+5,j, i+5, j);
                                                tot -= 32;
                                          }
                                          else {
                                                g.setColor(Color.black);
```

```
        g.drawLine(i+5, j, i+5, j);
        g.setColor(Color.white);
}
if(tot >= 16) {
        g.drawLine(i+4,j, i+4, j);
        tot -= 16;
}
else {
        g.setColor(Color.black);
        g.drawLine(i+4, j, i+4, j);
        g.setColor(Color.white);
}
if(tot >= 8) {
        g.drawLine(i+3,j, i+3, j);
        tot -= 8;
}
else {
        g.setColor(Color.black);
        g.drawLine(i+3, j, i+3, j);
        g.setColor(Color.white);
}
if(tot >= 4) {
        g.drawLine(i+2,j, i+2, j);
        tot -= 4;
}
else {
        g.setColor(Color.black);
        g.drawLine(i+2, j, i+2, j);
        g.setColor(Color.white);
}
if(tot >= 2) {
        g.drawLine(i+1,j, i+1, j);
        tot -= 2;
}
else {
        g.setColor(Color.black);
        g.drawLine(i+1, j, i+1, j);
        g.setColor(Color.white);
```

```
                    }
                    if(tot >= 1) {
                        g.drawLine(i,j, i, j);
                        tot -= 1;
                    }
                    else {
                        g.setColor(Color.black);
                        g.drawLine(i, j, i, j);
                        g.setColor(Color.white);
                    }

                }
            }
            br.close();
        }catch(Exception e){}
        }
 }

    public void read() {
        read = true;
        repaint();
    }

}
class PlayMP3Thread extends Thread {
    private URL url;
    public PlayMP3Thread(File mp3) {
        try {
            this.url = mp3.toURL();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    Player player;
    public void run() {
        try {
            MediaLocator ml = new MediaLocator(url);
            player = Manager.createPlayer(ml);
```

```
                                        player.addControllerListener(
                                                new ControllerListener()
                                                {
                                                        public void controllerUpdate(ControllerEvent event)
{
                                                                if(event instanceof EndOfMediaEvent) {
                                                                        player.stop();
                                                                        player.close();
                                                                }
                                                        }
                                                }
                                        );
                                        player.realize();
                                        player.start();
                                } catch(Exception e) {
                                        e.printStackTrace();
                                }
                        }
                        public void stopPlay() {
                                player.stop();
                        }
                }
}
```