

3D Modeling Turntable

Team Supermodel

Ben Bahorich, *Mechanical Engineering*
John Cate, *Computational and Applied Mathematics*
Robert LiKamWa, *Electrical and Computer Engineering*

Advisors

Mark Embree, *Computational and Applied Mathematics*
Lin Zhong, *Electrical and Computer Engineering*

Supervisor

Gary Woods, *Electrical and Computer Engineering*

Senior Design 2010 Final Report

Submitted to the TI Analog Design Competition

May 7, 2010

3D Modeling Turntable
Team Supermodel

Table of Contents

I.	Introduction	3
II.	Design Strategy and Implementation	3
	a. Computational Algorithms	3
	b. Image Gathering	5
	c. Electronic Turntable-Computer Interface	5
	d. Mechanical Considerations	7
III.	Final Design	7
	a. Calibration	7
	b. 3D Modeling	7
IV.	Results and Testing	8
	a. Successes and Limitations	8
	b. Volumetric Testing	8
V.	Achieved Design Specifications	8
VI.	Summary and Recommendations	8
	Acknowledgment	9
	References	9
Appendix A: Electronics		
I.	3DMT Interface Overview	A1
II.	Electronic Components	A1
III.	Electronic Schematic	A2
IV.	PCB Layout	A2
V.	Gallery	A3
Appendix B: MSP430 Code		B1
Appendix C: MATLAB Algorithm Scripts		
I.	Silhouetting Script	C1
II.	Calibration Script	C1
III.	3D Modeling Scripts	C3
Appendix D: Cost Breakdown		D1
Appendix E: Gallery		E1
Texas Instruments Chip Usage		Last page

3D Modeling Turntable

Team Supermodel

Ben Bahorich, John Cate, Robert LiKamWa

Mechanical Engineering, Computational and Applied Mathematics, Electrical and Computer Engineering,
Rice University
Houston, TX, USA
supermodel@roblkw.com

Executive Summary — 3D model creation is an emerging technology and finds itself employed in daily use for architect renderings and video game development. We present a solution that can automatically create faithful digital 3D models from real life objects. Our objective is to create a low-cost, easy-to-use system that can accurately model 3D objects. We achieved this objective through the creation of our 3D Modeling Turntable (3DMT). Using a silhouetting algorithm and a computer-interfaced turntable, we are able to successfully capture the volumetric likeness of a large variety of real-life 3D objects. The 3DMT has limitations; it captures a convex hull of the object, not modeling concavities in the surfaces, and foreground extraction for the silhouette fails when the object contains a color that matches the color of the background. Further improvements could come with the adoption of more sophisticated algorithms.

I. INTRODUCTION

3D modeling is prevalent in several fields, ranging from architecture to video game development. However, most 3D modeling is done using complicated 3D software such as 3D Studio MAX, Google Sketchup, or SolidWorks. A solution that can accurately model real-life 3D objects would be of significant benefit to architects who want to render their wooden/cardboard models and game developers who want to add real-life detail into their video game levels. Additionally, a 3D modeler would prove useful to online retailers who wish to provide customers with a preview of how an object looks. A vendor can show how a pair of sunglasses look from all sides and even overlay the 3D model on a customer.

3D modelers exist, prevalently in the form of 3D laser scanners. However, these cost upwards of \$5,000, out of the price range of several otherwise interested customers. Of secondary prevalence, 3D optical scanners are used industrially for reverse engineering purposes, but these involve complex setups requiring markers and/or laser lines. These types of scanners also cost \$3,000 and higher.

By creating a low-cost alternative to the available expensive solutions, Team Supermodel aims to fill a gap

in the 3D-modeling market with our 3DMT. We sought to create a system that costs less than \$250 in materials, creates 3D models with over 80% volumetric accuracy, and is controlled by a simple, easy-to-use user interface.

Our system meets these design criteria, and produces 3D models in a standard OBJ file format. These OBJ files can be imported into several 3D software packages, including Blender, 3DSMax, and SolidWorks. We will discuss the design of our product and explain our volumetric accuracy testing.

II. DESIGN STRATEGY AND IMPLEMENTATION

Team Supermodel's design strategy focused on taking advantage of our team's multi-disciplinary nature and used elements of Computational and Applied Math (CAAM), Electrical and Computer Engineering (ECE), and Mechanical Engineering (MECH). While our CAAM team member championed the algorithmic portion of the design, the ECE team member focused on the electronics interface, and the MECH student worked on the functional and aesthetic underlying mechanical design.

Using this strategy, we approached the various problems associated with 3D model capture. We looked at the algorithm we were using and designed a robust system that could provide all physical information and control necessary to produce the 3D models. To achieve this task, we developed a computer-controlled turntable that used two webcams to extract image data to form the 3D models. We wished to make the software easy-to-use from a user standpoint, yet still be fully capable of collecting data.

A. Computational Algorithms

There are three main algorithmic processes necessary for 3D model creation: Silhouetting, Calibration, and 3D Volume Reduction. The scripts for these can be found in Appendix C.

1) Silhouetting

In context of our project, silhouetting is the process of creating a bitmap of an image wherein any pixels in a foreground object are represented by 1's and every other pixel (i.e. the background pixels) are represented by 0's. We developed a simple threshold method to achieve the desired results. For a given image, a 'filter-point' pixel and a tolerance number between 0 and 255 is selected. Every pixel is then analyzed to determine if its red, green, and blue values (which range from 0-255) are all within the given tolerance of the filter-point. If a pixel is found to be within the tolerance, it is considered to be in the background, and assigned a value of 0 in the bitmap. If it is above or below the tolerance range, it is assigned a value of 1 in the bitmap. This method allows a user to select the filter-point in the background of an image, and, assuming proper lighting and consistent background color, the resulting bitmap will have 0's in place of background pixels and 1's in place of foreground pixels.

2) Calibration

In order to gather the desired 3D volumetric data, basic geometric information about each camera is required. We used Tsai's method of calibrating a camera with a single-view set of coplanar points^[1]. The method, in brief, is to solve a system of equations that is set up based on the observed image coordinates of a set of points and the real world position of those points using known equations for projections of 3D points to a 2D plane.

We designed a custom pattern of dots for use in the project. It began as 24 dots evenly spaced in a circle. One dot was designated the 'dominant' and made significantly larger than the others. A dot immediately adjacent to the 'dominant' was then removed. The resulting pattern can be seen in Fig. 1.

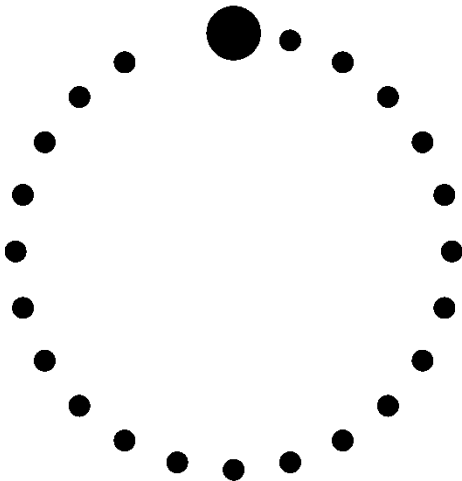


Figure 1. Calibration pattern

This pattern is placed on the turntable and images of it are captured at two different rotations for each camera. A silhouette of one of these images is created. The calibration software begins scanning through each pixel in the silhouette bitmap. When a 1 is found, indicating a dot in the image, a square box is created to surround it. The box is then expanded along one dimension by 1 pixel. This expansion continues as long as more 1's are added for each expansion. The box is then expanded along another dimension of the box, with the same stopping criteria. The resulting box encloses the entire dot as seen by the camera. The software then continues its scan of the image and the process is repeated with the next 1 that is found, if that pixel isn't already enclosed by another box. The resulting set of boxes encloses the dots as seen by the camera as seen in Fig. 2.



Figure 2. Boxes enclosing calibration pattern dots (colors inverted for ease of viewing)

The area and center point, in image coordinates, of each box is then calculated. The 'dominant' dot's box has a larger area than the other dots, by design. Using the center point as the dot's location, the Euclidean distances between the 'dominant' and all the other dots are calculated. The closest dot is considered as the next dot in the pattern. The distances from this dot to all the remaining dots are then calculated and the closest is, again, considered the next dot in the pattern. This process repeats until all 23 dots and their image coordinates are in 'pattern order'. The design of the calibration pattern is such that this order is always the same. The 'dominant' is easy for the software to recognize as it always appears larger to the camera, and thus has a larger rectangular box around it. By removing a dot immediately adjacent to the 'dominant', the software always traces around the pattern in the same direction, as the closest dot to the 'dominant' will always appear to be the one adjacent that was not removed.

With the knowledge of the image coordinates and the real world positions of the dots in the pattern, we are able to set up a system of equations to solve for the calibration information as laid out by Tsai^[1]. We get three parameters from this system: a 3x3 matrix R , a vector T , and a scalar focal distance f . The matrix and vector are used to take a point in real world coordinates and represent it in a coordinate system centered at the camera. Using this, we are able to work backwards and

find the real world coordinates of the origin and unit vectors in the x , y , and z directions of the camera coordinate system. The resulting vectors and points are used to define the position and orientation of the camera. The process is then repeated for the second image captured earlier. Note that it is taken by the same camera, but at a different table rotation angle. Using the orientations from the two different rotations, we are able to find the point around which the camera seems to have rotated (in reality it is the table that is rotating) and subsequently the rotational axis of the turntable. Combining the position, orientation, and focal distance of the camera with the knowledge of the orientation axis of the table, we are able to completely define a camera for the volume calculation algorithm.

3) 3D Volume Reduction

The basic premise of our 3D modeling algorithm is the reduction of a volume space based on what an object looks like from a number of perspectives.

We start with a large volume cube in a real world scale subdivided along the x and y axes based on the desired model resolution and stored as the top and bottom points of the resulting columns. A single frame is grabbed from a video of the object on the turntable and a silhouette of the image is produced. Using the geometric information of a camera and the rotational position of the turntable for the given image, we interpolate between the top and bottom points of a column and project the resulting full column from our volume cube into the image plane of the camera, as detailed by Niem^[2]. The resulting image coordinates represent an entire column of the volume cube as seen from the camera's perspective for the given silhouette.

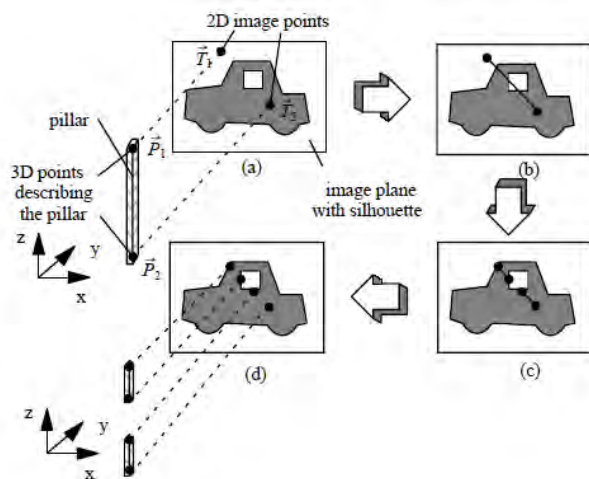


Figure 3. Column reduction and splitting^[1].

The algorithm checks each of the coordinates from the projected column and removes any that are not located inside the object silhouette. This is a simple check of whether the bitmap is a 1 or 0 at the image coordinate. The top and bottom of the reduced column are then stored. If a column is split in the reduction process (see Fig. 3), both of the resulting columns are stored. The process is repeated for each column in the volume cube. This will remove any pieces of the volume cube that are not in the object as seen from the perspective of the camera for the given frame of video.

Using the reduced volume instead of the entire volume cube, the entire process is repeated with the next desired frame. After a number of reductions, the resulting volume contains only points that were inside all of the object silhouettes from the different rotational positions of the turntable. This is the volumetric data of the object.

In our implementation, we used a number of rotational positions of the turntable as well as two separate cameras. This method required calibrating both cameras to synchronize the multi-view geometry, but did not change the implementation of the reduction algorithm.

B. Image Gathering

We decided to use a webcam to capture images so that we could drive down cost and easily pull image data as needed. Looking primarily for a low-cost webcam, we decided on using the Philips SPC 230NC webcam.

Upon initial testing of the algorithms using one webcam, we realized that one vantage point was not enough to discern crucial geometric information about an object. As such, we decided to use a secondary webcam pointed at a different angle to provide additional image information.

We decided to process 32 equally spaced turntable rotation angles for each camera, for a total of 64 different image frames.

Originally, the plan was to turn the motor to a specific angle, halt the turn, and take an image, but we decided that this still image process would take too long. Consequently, we decided instead to continuously rotate the table, acquire video streams, and pull out specific frames from the video streams in order to get the images we needed to run our algorithm.

We used a VLC command-line script to control the webcams and capture the video.

C. Electronic Turntable-Computer Interface

We used an Electronic Turntable-Computer interface to turn a lighting rig on and off, to turn a motor on and off, to control the motor's speed, and to track the angular position of the turntable's rotation.

1) *Microcontroller and Serial Line Driver*

The electronics component of the turntable was centered on a Texas Instruments MSP430, which provided adequate real-time capabilities to interface with the computer. We decided to use the MSP430F2012 as our microcontroller because it is easy to program and inexpensive to obtain.

The MSP430 acted as a slave to the computer—performing tasks when asked and reporting information as requested. In order to do this, we used a Serial Line Driver MAX3232 chip in order to interface the MSP430 with the serial port of a computer.

We used a Low Dropout voltage regulator to power both the MSP430 and the MAX3232 chip.

2) *Lighting Control*

The silhouetting algorithm required an evenly lit object and background, so we mounted LED strips on a tripod to provide such lighting. The 3DMT software controlled the lighting through the MSP430 by sending the character 'L' to the MSP430 to turn the lights on, and 'l' to turn the lights off. The MSP430 would then activate or deactivate a relay that controls the 12V of power being sent into the LED lighting strips.

3) *Motor Control*

The turntable's servomotor (further discussed below in *Mechanical Considerations*) was also controlled by the MSP430. The character 'M' was sent to the MSP430 to turn the motor on, and 'm' to turn the motor off. This would activate or deactivate a relay that sent 5V of power into the servomotor.

The MSP430 fed a Pulse Width Modulation (PWM) signal to the servomotor. The PWM signal is a train of pulses. The width of the pulse determines the speed of rotation of the table. The PWM made the table turn at a fairly constant speed—rotating the object at a continuous speed of rotation. To set the width of the pulse in the PWM, and thus the rotation, the character 'P' was sent to the MSP430, followed by a 4-digit number. The duty cycle of the PWM was then set as the 4-digit number. To amplify the PWM signal from the MSP430 to 5V, we used an NPN transistor in a common-emitter configuration.

4) *Angular Position Tracking*

The calibration and modeling algorithms required the precise angle rotation for each frame. Therefore, we designed an angle-tracking mechanism for this task (Fig. 4). First, a circular track was placed on the underside of the turntable, divided into 80 evenly spaced sectors, and colored in alternating white and black. Second, we set up an infrared LED and photodiode pair to watch the track. The reflectivity of the white sectors induced a high

current through the photodiode, while the black sectors induced a lower current. The signal was monitored by a comparator, which converted the photodiode analog signal to a digital signal. The digital signal was high when the LED was pointed at a white sector and low when the LED was pointed at a black sector. This signal was then fed to the MSP430 to count the sectors that go by. The MSP430 did this by counting the edges in the digital signal (i.e. when the signal goes from low to high or from high to low). When asked by the software by the serial command 'B', the MSP430 reported how many sectors went by to the software. The software asked for the count before and after 10 seconds and used that information to find the angular velocity of the turntable. With this, the software was able to determine the relative angular position of the turntable for each video frame. With lighting control, motor control, and angular position detection, the turntable was fully equipped to provide necessary information and control to the software.

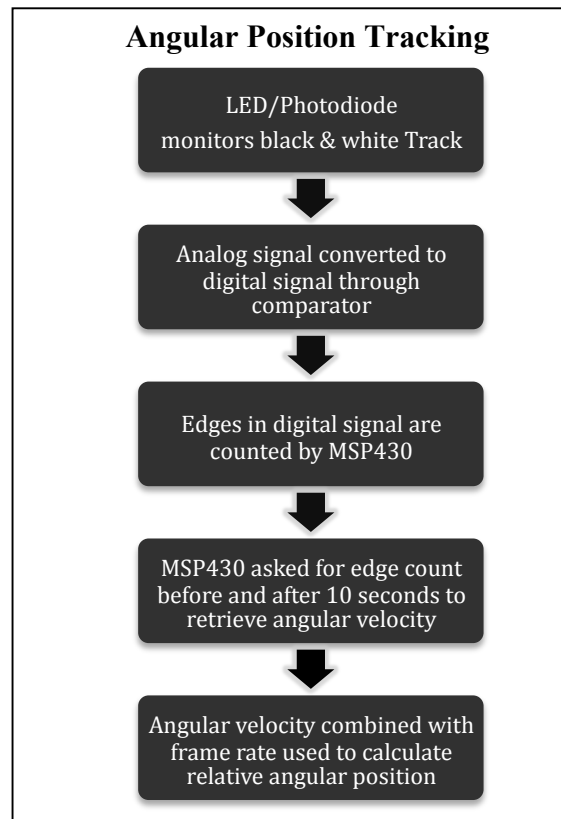


Figure 4. Angular Position Tracking

An electronic interface overview, a list of specific electronic components, an electronic schematic, and a PCB layout can be found in Appendix A. The MSP430 Software can be found in Appendix B.

D. Mechanical Considerations

Our main goal in designing the mechanical structures of the 3DMT was to create a system that would be simple and inexpensive. With this goal in mind, we decided to pursue a simple turntable setup as pictured in Fig. 5. As seen in Fig. 5, the system consists of a 17.5 inch acrylic platform rotated by a servomotor, a 20x20inch aluminum base supported by 4 rubber grommets, a cloth backdrop attached to the aluminum base, and 2 cameras and 3 LED bars mounted on a tripod constructed from aluminum and acrylic. The major mechanical considerations are discussed below.

We chose to use an S125 1T Sail Winch servo because it provided us with the torque we needed and the controllability we desired. With a simple pulse width modulation signal, we could control the rotation velocity. Furthermore, the S125 rotated our objects at a constant rate, which enabled us to run our algorithm using video capture rather than using still capture. Part of our design criteria was that our turntable would be able to rotate a 30lb object. With the S125 servo, we met this requirement.

Since our system was only designed to support loads up to 30lbs, in-depth mechanical strength analysis was unnecessary. However, we still performed some mechanical strength optimization. We used steel axels on our wheels to maximize strength and made the base out of aluminum to prevent flex.

We designed the turntable to be extremely stable by giving the base a large length and width (20x20 inches). We minimized constraining the acrylic platform by only using 3 wheels. In addition, we designed the tripod legs to extend 6 inches beyond the center of gravity thereby giving the tripod roughly a 40-degree tipping angle—that way the tripod would not fall over when bumped by the user.

One very important part of the mechanical design was the wheels. We needed to create wheels with very low friction because too much friction would prevent our servomotor from being able to rotate objects. We accomplished this by smoothing the contact surfaces and applying some oil lubricant. Friction was especially important in the wheels because they were placed towards the edge of the platform giving them a relatively large torque arm.

Another consideration was rotational inertia. We minimized the rotational inertia of the wheels and the platform by creating them out of lightweight materials.

The wheels were constructed out of common molding plastic and the platform was made from acrylic.

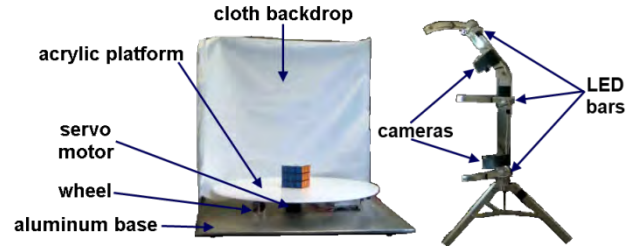


Figure 5. Physical Turntable Components

One of the main considerations in constructing our mechanical structures was aesthetics. We decided this was an important piece of our design because our clients would desire a system that appeared polished and refined. Therefore, we designed all of our mechanical pieces to complement each other visually. One of the reasons we employed a significant amount of acrylic in our design is that acrylic is visually appealing and professional.

III. FINAL DESIGN

Our final design cohesively pulled all of the design strategies together to provide an effective solution. The serial commands for the MSP430 were encoded in our MATLAB software script. Our software allows the user to easily perform the calibration and modeling phases.

A. Calibration

The user is asked to place the calibration pattern on the turntable. After doing so, the user then presses <Enter> on the computer. The lighting strips turn on and the table spins. The cameras gather two frames for each of the cameras. The table then stops turning.

The user defines a filter point for each frame of the gathered calibration images. The software determines camera position, rotation axis position, and focal length information and stores the information.

B. 3D Modeling

The user is asked to place the object to be modeled on the turntable. After doing so, the user then presses <Enter>. The lighting strips turn on and the table spins. As the table spins, the angular velocity is tracked, and the webcams capture several seconds of video. The table then stops turning. The user is asked to define a filter point for each webcam stream and the software runs the 3D modeling algorithm, creating the 3D model in a point cloud and mesh, and saving the 3D model in an OBJ file format.

IV. RESULTS AND TESTING

Our completed system is able to produce 3D models from images of real world objects. Some of the models we extracted can be found in our gallery in Appendix E. From a qualitative perspective, the models very closely resemble their real world counterparts. We were also able to do some quantitative accuracy testing to see just how well the system performed.

A. Successes and limitations

The turntable and software produces 3D models that look just like the real objects. For instance, we modeled a fedora and found that the resulting object included the crease on top of the hat. In addition, we ran a small head bust through the system and details such as ears, the nose, and even the eyes were clearly visible. For a low-resolution model, we found that the total process took no longer than 15 minutes from start to finish, including calibration.

The system is not without its limitations, however. We knew from the beginning that by utilizing silhouettes we would lose the ability to model concavities. For example, our system would not capture the inside of a cup. Through testing, we also found that background color provided another limitation. If part of an object is of a color similar to the background color, it is removed in the silhouetting process. This would result in the removal of parts of the reduced volume that should have been considered ‘in’ the object. Lastly, the 3D modeling algorithm can take several hours to generate a high-resolution model. On a positive note, this is several hours of the computer’s time, not the user’s time.

B. Volumetric Testing

We tested for volumetric accuracy only on models where we can compare the generated model against a completely accurate model. Therefore, we tested the Rubik’s Cube and the gaffer tape because we could create these simple models accurately in SolidWorks by simply extruding a solid of the correct dimensions. We call these solids the real object models.

After we create a solid of the correct dimensions in SolidWorks, we create a union (V_{Union}) of the two models (the real object extruded in SolidWorks and the model generated by the 3DMT) by anchoring both solids at a common center point. Then, we perform the calculations below to obtain the volumetric accuracy percentage (see Fig. 6 for reference).

$$\frac{V_{\text{UNION}} - V_{3\text{DMT}}}{V_{\text{RealObj}}} + \frac{V_{\text{UNION}} - V_{\text{RealObj}}}{V_{\text{RealObj}}} = \text{underlap}\% + \text{excess}\%$$

$$1 - (\text{underlap}\% + \text{excess}\%) = \text{Volumetric Accuracy } \%$$

We define the underlap as the portion of the real object that was missing in the 3DMT model. We define the excess as the portion of the 3DMT model that exceeded the volumetric bounds of the real object.

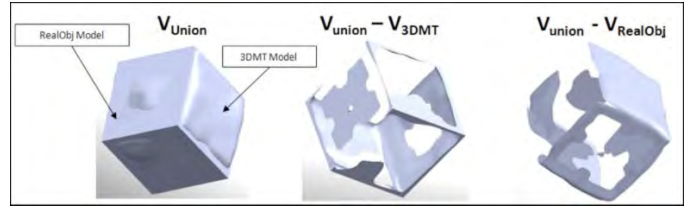


Figure 6. Left: Union of volumes, Center: Underlap, Right: Excess

The resulting volumetric accuracies both turned out to be over 90%, well above our initial design criterion of 80% volumetric accuracy. The volumetric accuracies can be found in Table I.

	RealObj cm ³	3DMT cm ³	UNION cm ³	Volumetric Accuracy cm ³
Rubik’s Cube	185.19	183.51	190.32	93.5%
Gaffer’s Tape	291.60	300.40	310.00	90.3%

V. ACHIEVED DESIGN SPECIFICATIONS

We met our initial design criteria, creating a solution that costs less than \$250 in materials (our cost breakdown is in Appendix D), provides 3D models with a volumetric accuracy of over 90%, and provides an easy-to-use user interface. Capturing data for generating a 3D model takes under 5 minutes. While generating the 3D model can take anywhere from 5 minutes to an hour, depending on the desired 3D resolution, it is important to note that this is not the user’s time, but the computer’s time. Several models can be run overnight if high resolutions are required for a particular project.

VI. SUMMARY AND RECOMMENDATIONS

Through the course of 9 months, Team Supermodel was able to utilize its expertise in Computational and Applied Mathematics, Electrical Engineering, and Mechanical Engineering to create a 3D modeling turntable. The 3DMT is able to capture 3D models of several different kinds of objects. We have shown that we can capture models with over a 90% volumetric accuracy. However, because the 3DMT employs a silhouetting algorithm, it is not able to capture concavities of surfaces. Furthermore, calibration can be a difficult procedure and could be improved by switching to a 3-dimensional calibration object rather than a plane of circles. To speed

up both the calibration and the 3D modeling phase, a future team should consider switching to C/C++ or Java.

The plan for the future is to propose the project to future senior design groups in the ECE and CAAM departments for potential software algorithm improvement. Robert LiKamWa will be pursuing a PhD at Rice University and will thus be in a prime position to act as a mentor for any future team.

The 3D Modeling Turntable has full video capture control, lighting control, motor control, and position tracking, all interfaced with a serial connection. The mechanical design has packaged the turntable in a functional and aesthetic form. The currently implemented algorithm provides a good basis for extracting 3D models, and the current design stands as a platform for future algorithmic development.

ACKNOWLEDGMENT

Team Supermodel would like to thank Lin Zhong and Mark Embree for advising us throughout the project. Additionally, Gary Woods provided essential support as the chief supervisor of our project. Monetary support was provided through the Electrical & Computer Engineering department at Rice University.

REFERENCES

- [1] R. Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses", *Journal of Robotics and Automation*, Vol. RA-3. No.4, August 1987, pp. 323-344.
- [2] Niem, Wolfgang. "Robust and Fast Modelling of 3D Natural Objects from Multiple Views." *SPIE Proceedings: Image and Video Processing II*. pp. 388-397, 1994

APPENDIX A: ELECTRONICS

APPENDIX A1: 3DMT INTERFACE OVERVIEW

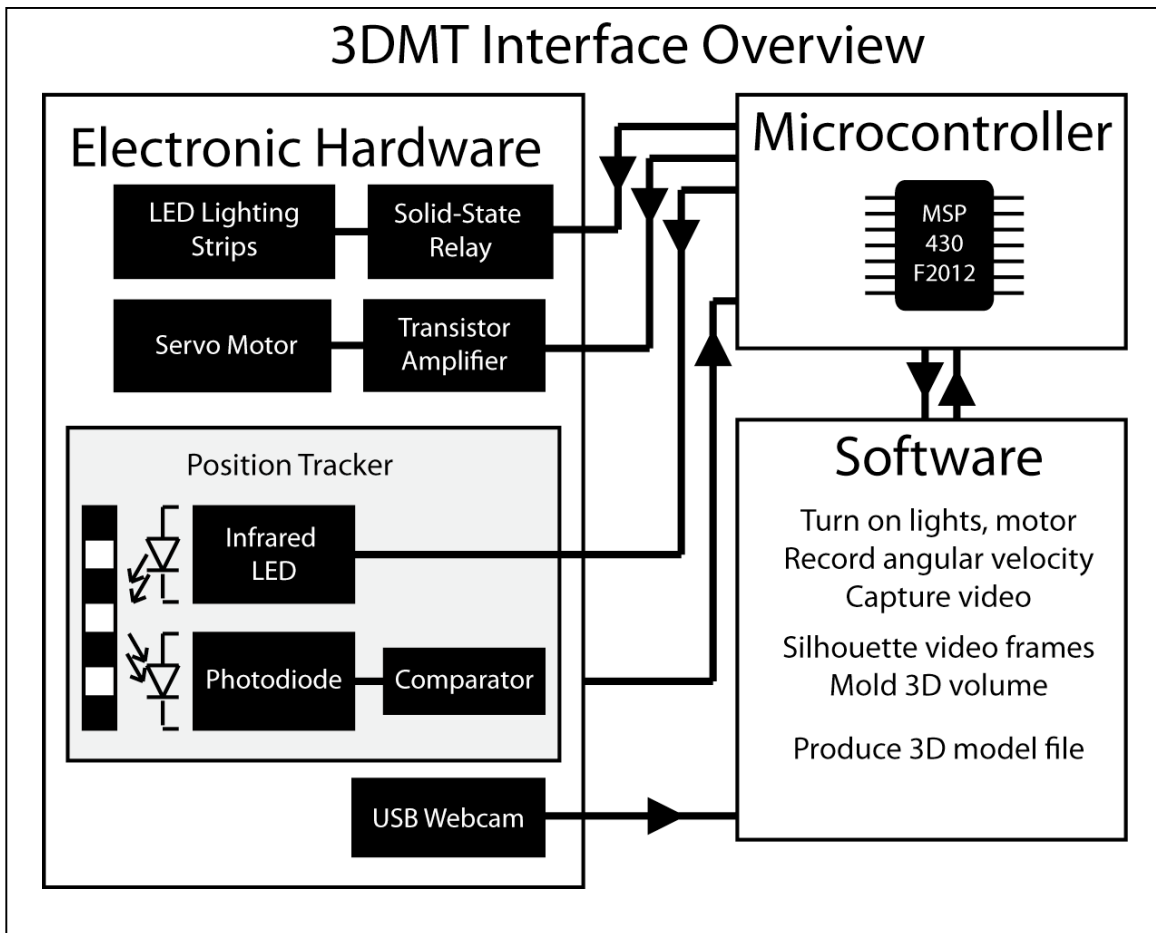


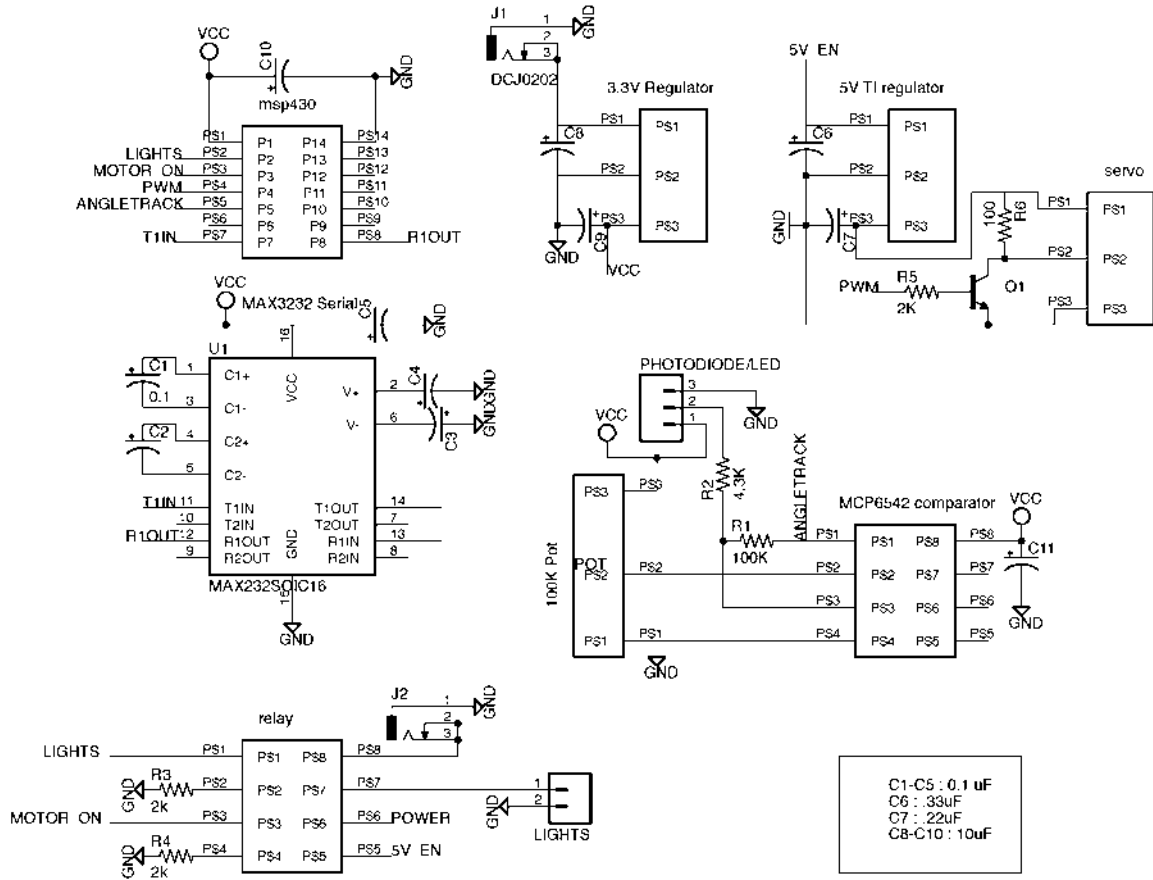
Figure A1: 3DMT Software-Turntable Interface

APPENDIX A2: ELECTRONIC COMPONENTS

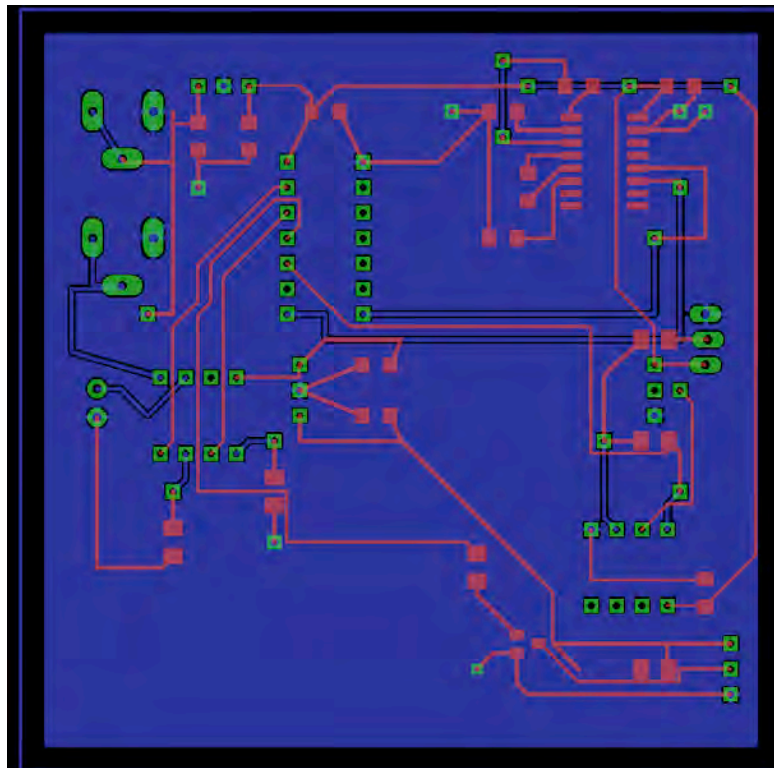
TABLE A2: ELECTRONIC COMPONENTS

	Electronic Components		
	Description	Part	Manufacturer
Microcontroller	Microcontroller	MSP430F2012	Texas Instruments
Serial Interface	3V to 5.5V Multichannel RS-232 Line Driver/Receiver	MAX3238E	
Voltage Regulator	5V Low-dropout	TL780	
	3.3V Low-dropout	UCC283T	
Comparator	Push-Pull Output	MCP6542	Microchip
Relay	Two-channel Solid State Relay	AQW212	Panasonic

APPENDIX A3: ELECTRONIC SCHEMATIC



APPENDIX A4: PCB LAYOUT



Blue is bottom. Red is top. Green is vias/holes.

APPENDIX D: COST OF MATERIALS

Cost Breakdown	
<i>Material</i>	<i>Cost</i>
Acrylic	\$50
Aluminum	\$75
Wheels	\$5
Circuitry	\$20
Webcams	\$35
Wire, Screws, etc.	\$5
Total	\$235

APPENDIX B: MSP430 TURNTABLE INTERFACE CODE

```

//*****
// MSP430 - Senior Design Turntable code
// Communicates via serial with computer software
// Turns on/off lights, motor
// Controls motor speed
// Counts and reports number of "blips" (edges in a digital signal) to estimate
// angular position of the turntable.
// Robert LiKamWa, Spring 2010
//*****

#include <msp430x20x2.h>

int receive();
void bittime();
void bittime15();
void send(int);
#define GRAY_IN 0x01 //in=0
#define LIGHTS_ON 0x02 //lighting rig on pin 1.1
#define LIGHTS_OFF 0xFD
#define PWM_BIT 0x04 //PWM pin is on pin 1.2
#define SIO_OUT_BIT 0x40//SIO_OUT pin 1.6
#define SIO_OUT_CLR 0xbf
#define SIO_INP_BIT 0x20//SIO_IN pin 1.5
#define MOTOR_ON 0x08//pin 1.3
#define MOTOR_OFF 0xF7//

int blips,blips_on;

int main(void) {
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    BCSCTL1 = CALBC1_8MHZ; // set clock to 8MHz
    DCOCTL = CALDCO_8MHZ;

    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    P1DIR |= PWM_BIT; // P1.2 and P1.3 output
    P1SEL |= PWM_BIT; // P1.2 and P1.3 TA1/2 options
    CCR0 = 65535; // PWM Period
    CCTL1 = OUTMOD_7; // CCR1 reset/set
    CCR1 = 15000; // CCR1 PWM duty cycle
    TACTL = TASSEL_2 + MC_1; // SMCLK, up mode
    P1DIR=0;
    P1DIR |= SIO_OUT_BIT;
    P1DIR |= LIGHTS_ON;
    P1DIR |= MOTOR_ON;

    P1OUT = 0; // clear OUT bits
    blips = 0; //count blips

    int c,c1,c2,c3,c4,temp;
    CCR1=5000; //set PWM (motor speed)
    P1OUT&=MOTOR_OFF; //start with motor off
    P1OUT&= LIGHTS_OFF; //start with lights off

    for (;){
        volatile unsigned int i,j; // volatile to prevent optimization
        P1OUT|=MOTOR_ON;

```

```

P1OUT |= LIGHTS_ON;

//if photodiode/comparator goes (1 -> 0) or (0 -> 1), blips is incremented
if (P1IN&GRAY_IN != 0){
    if (blips_on==0){
        blips++;
        blips%=10000;
        blips_on=1;
    }
} else{
    if (blips_on==1){
        blips++;
        blips%=10000;
        blips_on=0;
    }
}

c=receive();//get character
if (c=='P'){//P - motor speed
    c1=receive();//get 4 digit PWM number
    c1-='0';
    c2=receive();//get 4 digit PWM number
    c2-='0';
c3=receive();//get 4 digit PWM number
c3-='0';
c4=receive();//get 4 digit PWM number
c4-='0';
    temp=c4;
    temp+=c3*10;
    temp+=c2*100;
    temp+=c1*1000;
    CCR1=temp*10;//set the PWM
} else if(c=='M'){//M - motor on
P1OUT|=MOTOR_ON;
} else if(c=='m'){//m - motor off
P1OUT&=MOTOR_OFF;
} else if (c=='L'){//L - lights on
P1OUT |= LIGHTS_ON;
} else if (c=='l'){//l - lights off
P1OUT &= LIGHTS_OFF;
} else if (c=='B'){//B - echo blips to the computer, then clear the blips
P1OUT |= LIGHTS_ON;
c1=blips/1000;
c2=(blips-c1*1000)/100;
c3=(blips-c2*100-c1*1000)/10;
c4=blips-c1*1000-c2*100-c3*10;
send(c1+'0');
send(c2+'0');
send(c3+'0');
send(c4+'0');
    blips=0;
}
}
}

```

```

void send(int c) {
    int i=8;
    P1OUT &= SIO_OUT_CLR; // begin the start bit, 0

```

```

    bittime();
    do{
        if(c&1)
            P1OUT |= SIO_OUT_BIT;
        else
            P1OUT &= SIO_OUT_CLR;
        c>>=1;
        bittime();
    }while (--i != 0);
    P1OUT |= SIO_OUT_BIT; // begin the stop bit
    bittime();
    bittime(); // two stop bits
    P1OUT &= SIO_RDY_CLR;
}

int receive() {
    int i=8, c=0;
    // wait for the start bit. while waiting, increment blips if necessary
    while(P1IN & SIO_INP_BIT){
        if (P1IN&GRAY_IN != 0){
            if (blips_on==0){
                blips++;
                blips%=1000;
                blips_on=1;
            }
        }else{
            if (blips_on==1){
                blips++;
                blips%=1000;
                blips_on=0;
            }
        }
    }
    // delay till the middle of the first data bit
    bittime15();

    // collect the next 8 bits
    do{
        c>>=1;
        if(P1IN & SIO_INP_BIT)
            c |= 0x80;
        bittime();
    }while (--i != 0);
    return(c);
}

void bittime() {
    volatile unsigned int i=77;
    do{i--;} while(i!=0);
}

void bittime15() {
    volatile unsigned int i=115;
    do{i--;} while(i!=0);
}

```

APPENDIX C: MATLAB ALGORITHM SCRIPTS

(Only representative scripts are shown here. All scripts are available upon request)

APPENDIX C1: SILHOUETTING SCRIPT

```
function iff = sill(img,point,tol,crop,gauss)

    imgm = img;
    if(gauss==1)
        imgm = imfilter(img,fspecial('gaussian',10,10)); %imread(img);
    end

    [imrow, imcol] = size(imgm);
    id = double(imgm);
    cf = double(point);

    if1 = double(id(:,:,1)>(cf(1)-tol)) + double(id(:,:,1)<(cf(1)+tol));
    if2 = double(id(:,:,2)>(cf(2)-tol)) + double(id(:,:,2)<(cf(2)+tol));
    if3 = double(id(:,:,3)>(cf(3)-tol)) + double(id(:,:,3)<(cf(3)+tol));

    iff = (if1+if2+if3)==6;

    %cropping
    if(~isempty(crop))
        iff(1:crop(1),:) = 1;      %top
        iff(crop(2):end,:) = 1;    %bottom
        iff(:,1:crop(3)) = 1;      %left
        iff(:,crop(4):end) = 1;    %right
    end
```

APPENDIX C2: CALIBRATION SCRIPT

```
function [hp0,vp0, focal, pivot,f,Points] = calibrate(img1,img2)
    0 = [];
    Xn = [];
    Yn = [];
    Zn = [];

    imgs = {img1, img2};

    for index = [1 2]
        img = imgs{index};

        [t p crop gauss] = GUI_3_22_10(img);% pulls calibration image
        XY = [];

        sil = sill(img,p,t,crop,gauss); %silhouettes the image

        sil = 1-sil;
        figure(1)
        if(size(XY,2)==0)
            disp('retrieving calibration points')
            [XY, Points] = pointExtractDot(sil,1);
        end
        close(1)
        Points = [Points;zeros(1,size(Points,2))];
    end
```

```

xw = Points(1,:);
yw = Points(2,:);
zw = Points(3,:);

%calibrate
Xf = XY(1,:);
Yf = XY(2,:);

Cx = (size(sil,2)+1)/2;    %center column
Cy = (size(sil,1)+1)/2;

sW = 4.536*10;
sH = 3.416*10;
ccdArea = sW*sH;
dx = sqrt(ccdArea/1300000);
dy = dx;
Ncx = sW/dx;
Ncy = sH/dy;
Nfx=size(sil,2);
dxPrime = dx*Ncx/Nfx;
Nfy=size(sil,1);
dyPrime = dy*Ncy/Nfy;
sx=1;

Xd = sx^-1*dxPrime.*(Xf-Cx);
Yd = dyPrime.*(Yf-Cy);

T = [Yd'.*xw' Yd'.*yw' Yd' -Xd'.*xw' -Xd'.*yw']\Xd';

r1p = T(1);r2p = T(2); r4p = T(4); r5p = T(5);
Sr = r1p^2+r2p^2+r4p^2+r5p^2;
Ty = sqrt((Sr - sqrt((Sr^2-4*(r1p*r5p-r4p*r2p)^2)))/(2*(r1p*r5p-r4p*r2p)^2));

%Sign of Ty trial
i=8;
xwtemp = xw(i);
ywtemp = yw(i);
Ty = abs(Ty);
r1 = (r1p)*Ty; r2 = (r2p)*Ty; r4 = (r4p)*Ty; r5 = (r5p)*Ty; Tx = T(3)*Ty;

x = r1*xwtemp+r2*ywtemp+Tx;
y = r4*xwtemp+r5*ywtemp+Ty;

X = Xd*Nfx/(dx*Ncx);
Y = (Yf-Cy);    %need to get coordinate of picture, not matrix entries

if(sign(Xd(i))~=sign(x) || sign(Yd(i))~=sign(y))
    Ty = -Ty;
end

%compute 3D rotation matrix
r1 = (r1p)*Ty; r2 = (r2p)*Ty; r4 = (r4p)*Ty; r5 = (r5p)*Ty; Tx = T(3)*Ty;
s = -sign(r1*r4+r2*r5);
R = [r1 r2 (1-r1^2-r2^2)^(1/2);
     r4 r5 s*(1-r4^2-r5^2)^(1/2)];
R = [R;cross(R(1,:),R(2,:))];

r7 = R(3,1); r8 = R(3,2); r9 = R(3,3);

```

```

y=zeros(length(Xd),1);
w=zeros(length(Xd),1);
for i=1:length(Xd);
    y(i) = r4*xw(i)+r5*yw(i) + Ty;
    w(i) = r7*xw(i)+r8*yw(i)+r9*0;
end

fT = [y -dy*Y']\ (w*dy.*Y');
Tz = fT(2);
f = fT(1);
T = [Tx;Ty;Tz];

if(f<0)
    R = [r1 r2 -R(1,3);
         r4 r5 -R(2,3);
         -R(3,1) -R(3,2) R(3,3)];

    y=zeros(length(Xd),1);
    w=zeros(length(Xd),1);
    for i=1:length(Xd);
        y(i) = R(2,1)*xw(i)+R(2,2)*yw(i) + Ty;
        w(i) = R(3,1)*xw(i)+R(3,2)*yw(i)+r9*0;
    end

    fT = [y -dy*Y']\ (w*dy.*Y');
    Tz = fT(2);
    f = fT(1);
    T = [Tx;Ty;Tz];
end
O = [0 R'*([0;0;0]-T)];
Xn = [Xn R'*([1;0;0]-T)];
Yn = [Yn R'*([0;1;0]-T)];
Zn = [Zn R'*([0;0;1]-T)];
f = f/dy;
Xu = [];
Yu = [];
end

Z = Zn-0;Y = Yn-0;X = Xn-0;
c = [R'*([0;0;2]-T)];
focal = 0(:,1);
n0 = Zn(:,1)-0(:,1);
u0 = [0 sqrt(n0(1)^2+n0(2)^2) n0(3)]';
hp0 = Xn(:,1)-0(:,1);
vp0 = Yn(:,1)-0(:,1);
pivot = focal + n0;

```

APPENDIX C3: 3D MODELING SCRIPTS

```

function nPts = computeVolume(calParams,AnglesPerSecond)
    hp10=calParams.hp10;
    vp10=calParams.vp10;
    focal1=calParams.focal1;
    pivot1=calParams.pivot1;
    f1=calParams.f1;
    hp20=calParams.hp20;
    vp20=calParams.vp20;
    focal2=calParams.focal2;
    pivot2=calParams.pivot2;
    f2=calParams.f2;

```

```

sf = 60;
totalTime = (2*pi)/AnglesPerSecond;
totalFrames = totalTime*30;

n = 32;

frmArray = sf:(totalFrames/n):sf+totalFrames;
angArray = 0:(2*pi)/(length(frmArray)-1):2*pi;
frmArray = frmArray(1:end-1);
angArray = angArray(1:end-1);

disp('retrieving frames, video1...')
[video1, audio1] = mmread('test.mpg', frmArray);

disp('retrieving frames, video2...')
[video2, audio2] = mmread('test2.mpg', frmArray);

clear audio1 audio2
[t1 p1 crop1 gauss1] = GUI_3_22_10(video1.frames(1).cdata);
[t2 p2 crop2 gauss2] = GUI_3_22_10(video2.frames(1).cdata);

disp('Computing Volume...')
Pts = [];
angArray = 2*pi.*(video1.times./totalTime);
for i=1:length(frmArray)
    disp(['Frame ' num2str(i) ' of ' num2str(length(frmArray)*2) '...'])
    Pts = to3DPts(video1.frames(i).cdata, Pts, hp10, vp10, angArray(i),
                  focal1, pivot1, f1, t1, p1, crop1, gauss1);
end

for i=1:length(frmArray)
    disp(['Frame ' num2str(i+length(frmArray)) ' of '
          num2str(length(frmArray)*2) '...'])
    Pts = to3DPts(video2.frames(i).cdata, Pts, hp20, vp20, angArray(i),
                  focal2, pivot2, f2, t2, p2, crop2, gauss2);
end
disp('Computing Volume... DONE')
nPts = [Pts(1:3,:) Pts(4:6,:)];

function keep = to3DPts(img, Pts, hp0, vp0, thetax, focal, pivot, f, tol, p, crop, gauss)
% Pts - 6x: vector of points describing columns - rows 1:3 are point 1, 4:6
% are point 2 in XYZ order.
% thetax - rotation about x axis
% thetax - rotation about z axis

center = [10 15 0]';

sil = 1 - sill(img, p, tol, crop, gauss); thetax = -thetaz;
Rz = [cos(thetaz) -sin(thetaz) 0; sin(thetaz) cos(thetaz) 0; 0 0 1]; %rotation
matrix for z axis

pivot = Rz*(pivot - center)+center;
c=Rz*(focal-center)+center;
a = c-pivot;
a = -a/norm(a);

hp = (Rz)*hp0; %'h' - unit vector parallel to horizontal axis of image plane in WCS
vp = (Rz)*vp0; %'v' - unit vector parallel to vertical axis of image plane in WCS

```

```

%calibrate Xh, Yh
Pp = project(pivot,c,a,f*hp,f*vp,0,0);
Xh = size(sil,2)/2-Pp(1);
Yh = size(sil,1)/2-Pp(2);
h = f*hp+a*Xh;
v = f*vp+a*Yh;

ex = 15;
if(isempty(Pts))
    Pts = generatePts([center(1)-ex,center(1)+ex],[center(2)-
ex,center(2)+ex],[center(3),center(3)+ex+5],40);
end
nlim = 100000;
keep = zeros(6,nlim);

n=0;
T = [];
B2 = [];
B = zeros(2,100000);
n2=1;
n2lim = 100000;
hold on

for k = 1:size(Pts,2)

    P1 = Pts(1:3,k); %pull top point of column from Pts data structure
    P2 = Pts(4:6,k); %pull bottom point of column from Pts data structure

    T1 = project(P1,c,a,h,v,Xh,Yh);    %T1 = (X,Y)
    T2 = project(P2,c,a,h,v,Xh,Yh);

    top = T1;          %bottom y value of pillar (rounded for image coordinates)
    bottom = T2;      %top y value of pillar (rounded for image coordinates)

    diff=1;
    dx = 1;
    while(diff>=1)
        np = pillarLine(P1,P2,dx);
        dT2 = project(np,c,a,h,v,Xh,Yh);

        diff = max(abs(top-dT2));
        dx = dx/2;
    end
    dx = dx*2;
    minz=[];          %used to figure out when a reduced column stops
    maxz=[];          %used to figure out when a reduced column starts

    for i = 0:dx:1

        pillarPoint = P1+i*(P2-P1);%pillarLine(P1,P2,i);
        T3 = project(pillarPoint,c,a,h,v,Xh,Yh); %[I3+Xh;J3+Yh]; %formula as in
paper    %T1 = (X,Y)

        y = round(T3(2));
        x = round(T3(1));
        C = x;
        R = y;
        if(C>0 && C<641 && R>0 && R<481)
            if(sil(R,C)==1)
                if(isempty(maxz));

```

```

        maxz = [x;y;pillarPoint];
    end
    minz = [x;y;pillarPoint];

    if(n>nlim)
        nlim = nlim+100000;
        disp('expand')
        keep = [keep zeros(6,100000)];
    end

    else if(~isempty(maxz))
        n = n+1;
        keep(:,n) = [maxz(3:end);minz(3:end)];
        maxz = [];
        minz = [];

        end
    end
    if(~isempty(maxz) && i==1)
        n = n+1;
        keep(:,n) = [maxz(3:end);minz(3:end)];

        maxz = [];
        minz = [];

        end
    end
end
end

keep = keep(:,1:n);

function P = pillarLine(top,bottom,t)

    P = top+t*(bottom-top);

    function T1 = project(P1,c,a,h,v,Xh,Yh)
        I1 = ((P1-c)*h)/((P1-c)*a); %formula as in paper
        J1 = ((P1-c)*v)/((P1-c)*a); %formula as in paper
        T1 = [I1;J1]; %formula as in paper    %T1 = (X,Y)

    function pts = generatePts(x,y,z,width)
        pts = zeros(6,width^2);
        X = x(1):(x(end)-x(1))/(width-1):x(end);    %X coordinate span
        Y = y(1):(y(end)-y(1))/(width-1):y(end);    %Y coordinate span
        n = 1;
        for i = 1:width
            for j = 1:width
                pts(:,n) = [X(i) Y(j) z(2) X(i) Y(j) z(1)]';    %column vector of two
                triplet points
                n = n+1;
            end
        end
        disp('points generated')

    function [R,C] = xy2matrix(x,y,Height)

        C = 640 - (x+1);
        R = y;

```

Appendix E: 3D Modeling Turntable Gallery

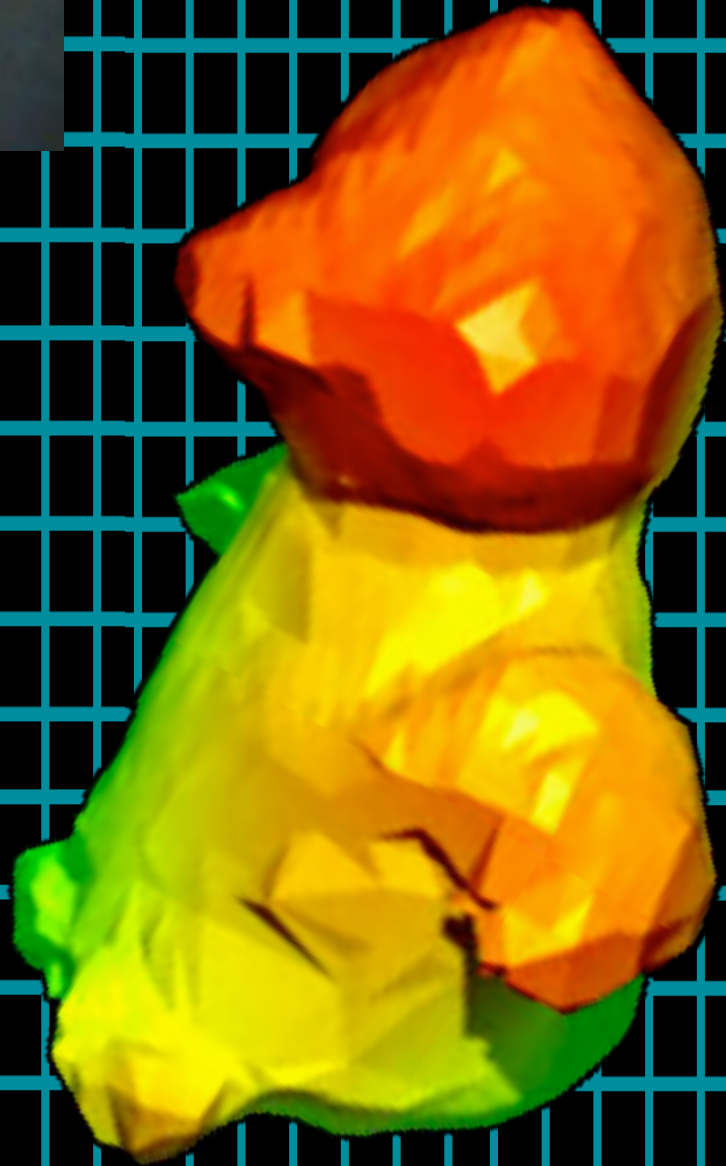
This gallery is best viewed with 3D
Chromadepth glasses, such as Crayola
3D sidewalk chalk glasses

Images are also viewable without such
glasses.

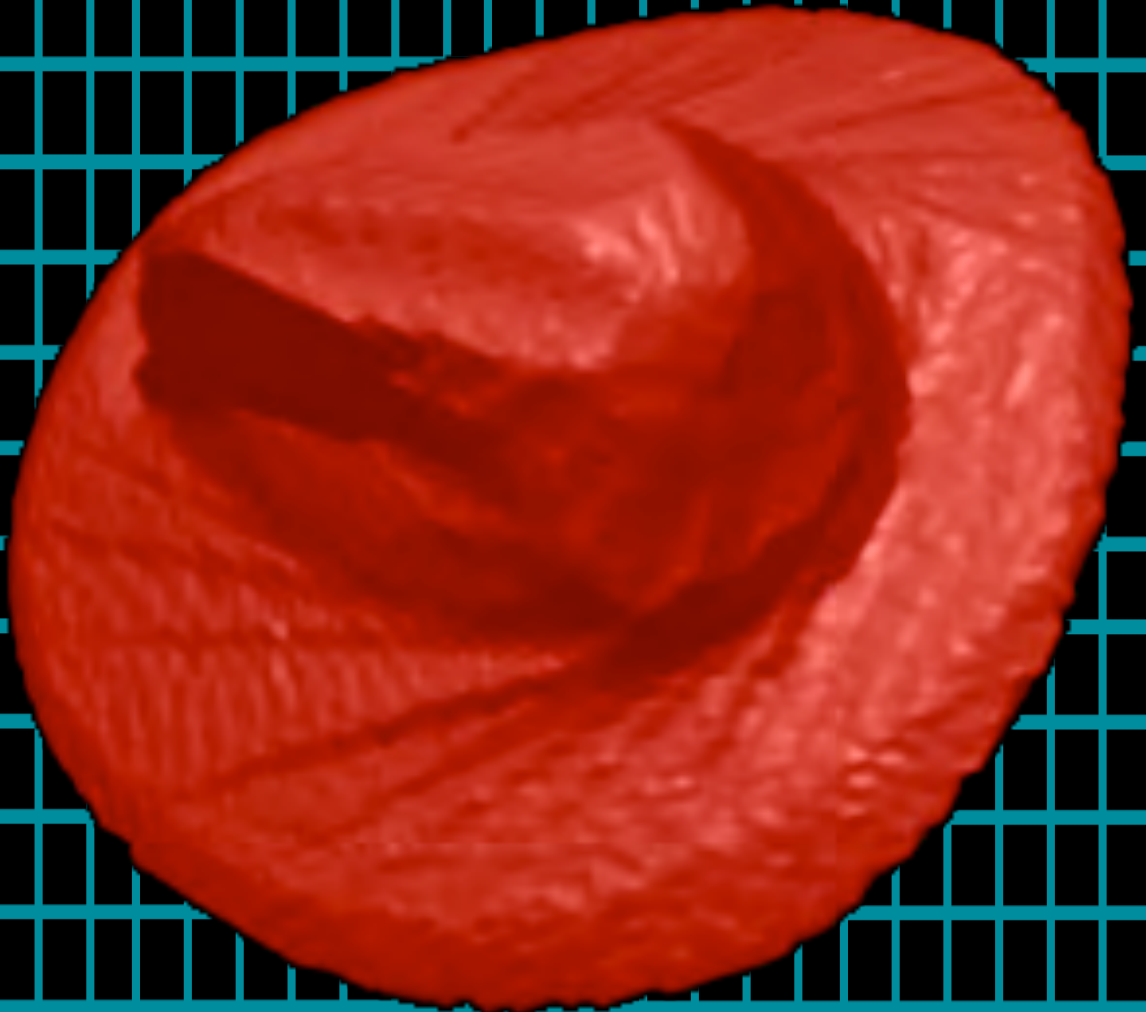
Willy's Statue



Ducky

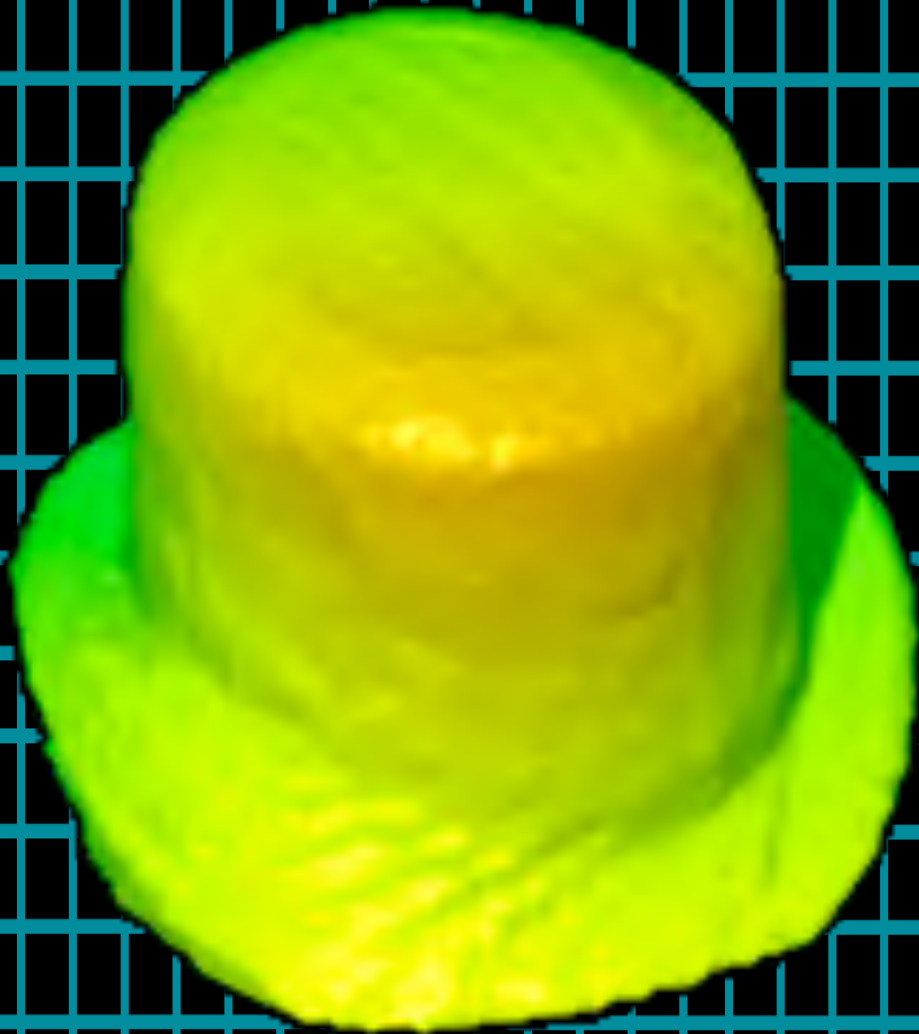


Fedora

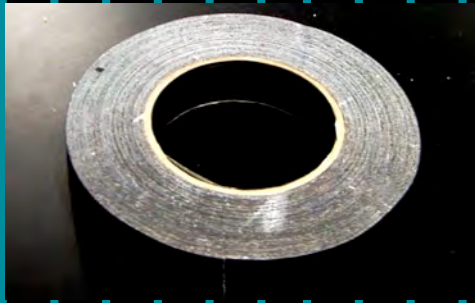




Top Hat

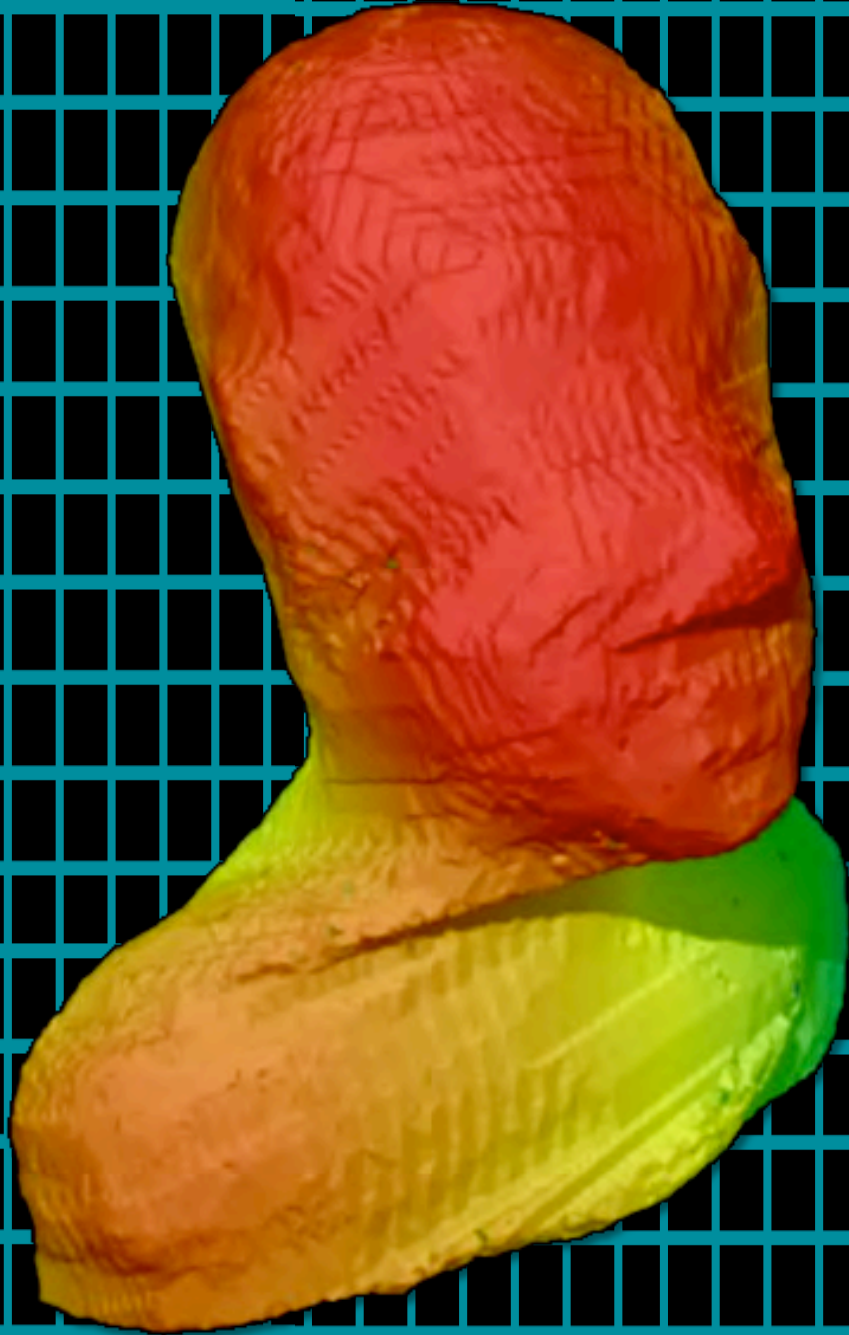


Gaffer's Tape

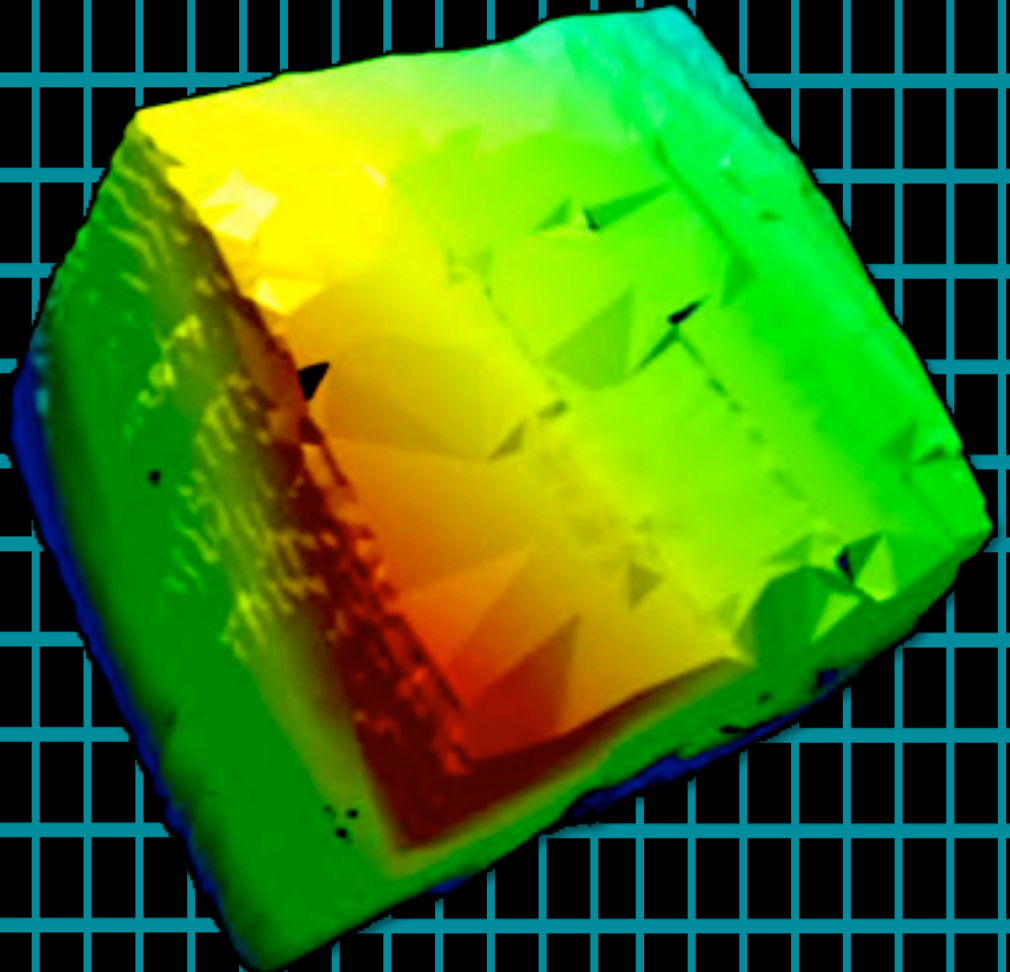
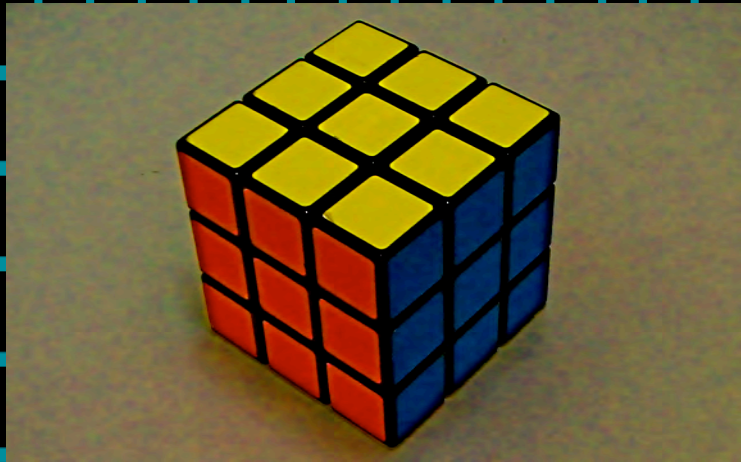


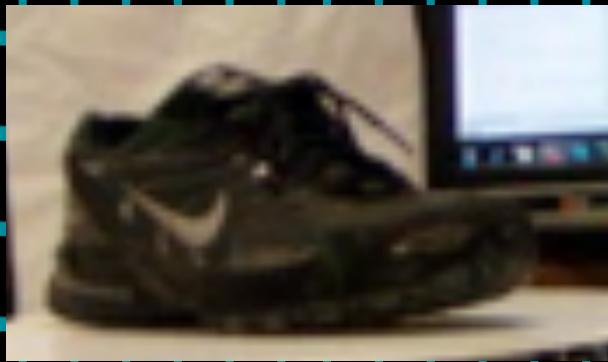


Head Bust

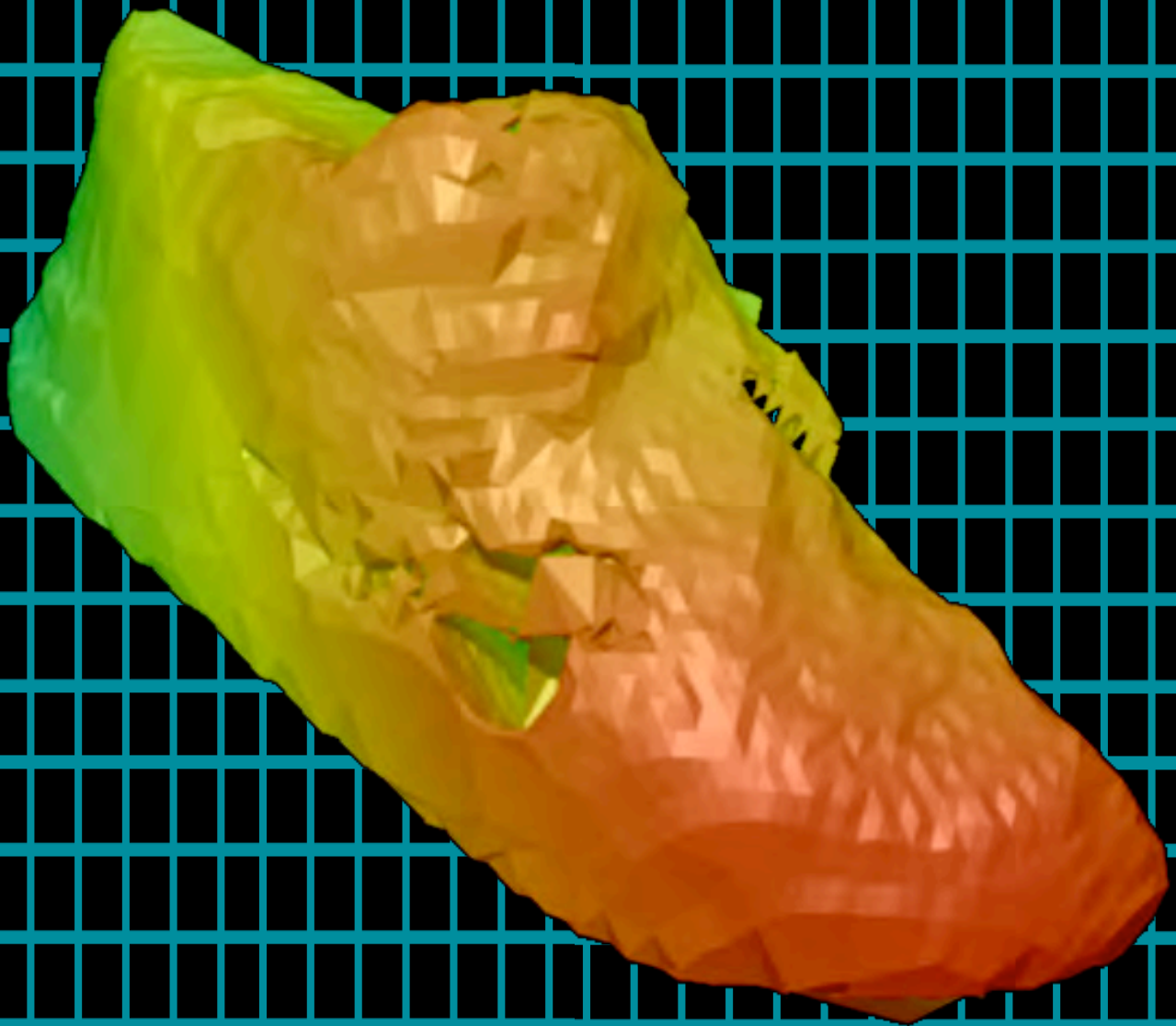


Rubik's Cube





Shoe



Texas Instruments Chip Usage

MSP430F2012 Microcontroller:

Our electronic interface was centered on the MSP430F2012 microcontroller. We used its digital input and output pins to handle the serial connection, control the lights, control the motor, and read the Photodiode/LED position tracker. The MSP430F2012 also had the Pulse Width Modulation capability necessary to control the speed of our servomotor.

The chief differentiator between the MSP430F2012 and other microcontrollers was its ease-of-use. The instruction set of the MSP430 is simple and intuitive, and the simplicity of the Spy-Bi-Wire interface allows us to download and debug our MSP430 quickly and easily. We use IAR Embedded Workbench to create and download our programs.

MAX3232E Serial Line Driver:

The MAX3232E acted as our Serial Line Driver, enabling communication between the MSP430 and the computer. The Line Driver was necessary to reduce the high serial voltage to one readable by our microcontroller. We chose to use the MAX3232E because it can convert the serial voltages to the 3V that is readable for the low-power MSP430 logic.

TL780 5V Low Dropout Regulator:

The TL780 converted our AC wall wart's 9V power supply to provide the servo with a constant 5V of power. We chose the TL780 because it provides 1.5A of power, enough to drive our servomotor at a very high torque. We also chose the chip out of its simplicity to implement.

UCC283T-3 3.3V Low Dropout Regulator:

The UCC283T-3 converted our AC wall wart's 9V power supply to provide the MSP430 with a constant 3.3V of power. It also powered the LED and photodiode pair that tracked the position of the turntable. We also chose the chip out of its simplicity to implement.