Recovering 3D Geometry with a Robot Arm and a Laser Scanner

James Kuffner, Jr
CS 223b : Computer Vision
Prof. Carlo Tomasi

March 21, 1995

Contents

1 Introduction

The goal of this project is to reconstruct the 3D geometry of a static object. Given a robot arm that emits a plane of laser light, the idea is to sweep the plane of light across the surface of the object and view the profile of the object contour using an off-axis CCD camera mounted near the laser. The resulting camera images can be used to compute the 3D coordinates of points on the object surface via triangulation.

The three main components of the 3D geometry recovery system are the camera image interpretation, mesh generation, and visualization software. Each section below contains a description of the component's functionality, the theory or mathematics underlying my implementation, and a discussion of the experimental results obtained. The actual code listings for each software component are included in Appendix A through Appendix C.

2 Hardware and Data Acquisition

The anticipated primary source of data for this project was a Nomadic, Inc. NOMAD2 mobile robot[1]. The robot has a device that emits a plane of monochromatic laser light mounted on top of its exterior, along with an off-axis color CCD camera. The laser light is projected out into the robot's environment, and a profile "stripe" is seen by the camera, which is equipped with a filter that permits only light of roughly the same frequency as the laser.

The robot control system allows the user to specify a trajectory for the robot to follow, as it has wheels that allow it to translate and rotate arbitrarily. Given a stationary object in the world, the robot can follow a trajectory around the object while recording the laser stripe images seen at each point. The 3D geometry of points on the object surface can then be reconstructed by taking into account the position and orientation of the robot, as well as the camera image seen at each point along the trajectory. For reasons having to do with the accuracy of control and the simplicity of the calculations, the robot is usually given a trajectory comprised of a straight line parallel to the axis along which the object's geometry will be scanned.

This was my original idea for data collection. Unfortunately, the robot's navigation system was scheduled to be upgraded during the time I planned to acquire data. Nomadic, Inc. has to perform both the hardware and software upgrades, and they were scheduled to do so during mid-February. However, they have yet to complete the upgrades at the time of the writing of this report.

Since I was unable to wait around for the delinquent upgrades, I managed to acquire a sequence of laser range-finding data from a CyberWare scanner[2]. The CyberWare scanner is essentially the same setup as the robot, except that the laser and camera are stationary,

[1] the robot is located in Laurel Hall in the Robotics Trailer complex.

[2] the scanner belongs to Professor Marc Levoy's research group and is located in the CIS graphics lab.

and the object being scanned rides atop a movable platform perpendicular to the direction of laser emission. The platform's position can be controlled quite accurately, resulting in potentially very precise geometry measurements.

Using some sample data of a model bunny acquired by the CyberWare scanner, I was able to write all of the software for my project and test it. The results of this experiment are what comprises the bulk of this report. There were 170 greyscale images of size 124 pixels wide by 121 pixels high used in the geometry computation. Each pixel contains a value proportional to the intensity of the laser light perceived at the corresponding point on the camera sensor. Thirty-two sample images of the raw camera data of the bunny are shown in Figure 1 on the following page.

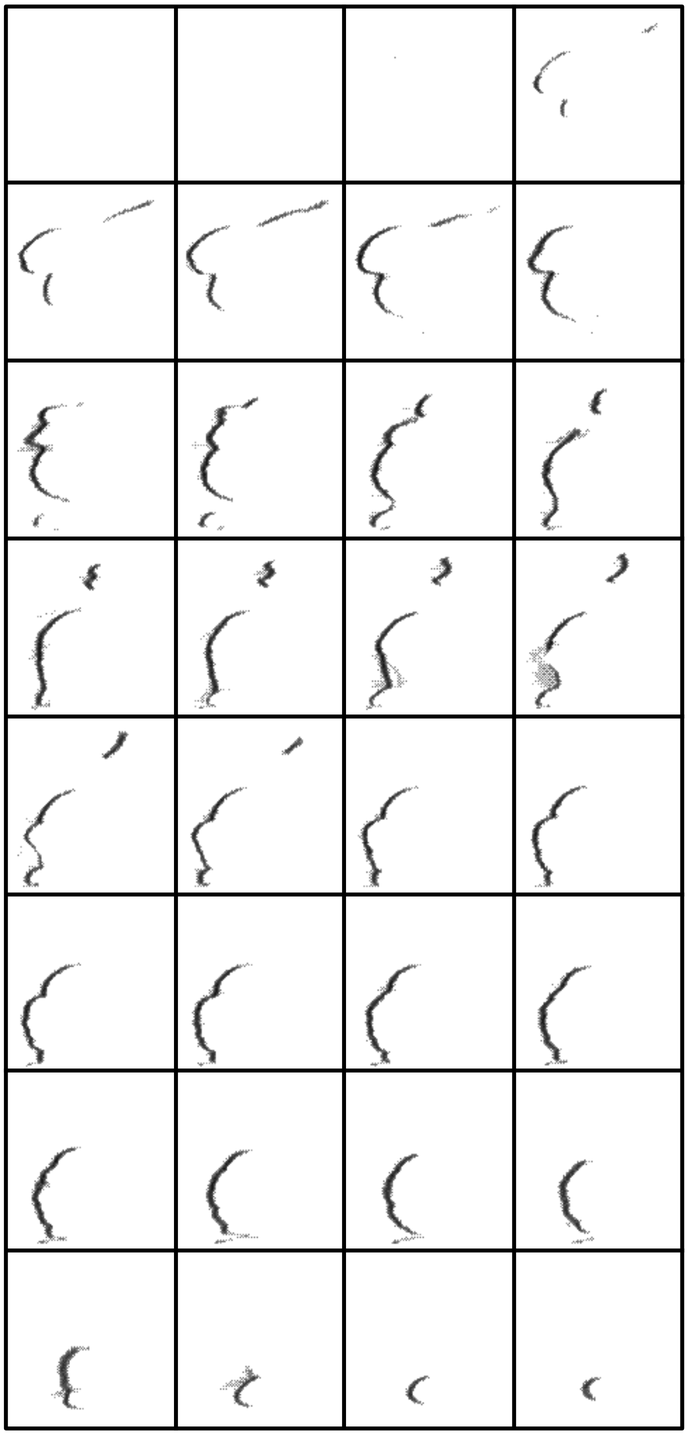

Figure 1. A sequence of raw camera data images for the model bunny. These 32 scan images show the laser stripe seen by the camera for a given position, proceeding from the top-left to the bottom-right image. *NOTE: for clarity in reproduction, these images are shown inverted. (black-on-white instead of white-on-black).*

3 Camera Image Interpretation

3.1 Description

The goal of this component is to convert the sequence of camera images representing different "contours" of the viewed object into a set of 3D points. The image interpretation consists of two phases:

1. A calculation to correct for camera image distortion (and radiometric dropoff if necessary).
2. A program to triangulate points along the laser stripe seen in each image and to output 3D coordinates of the object surface.

The first phase (correcting for distortion) I decided not to implement. Three reasons motivated this decision. First, the CyberWare equipment that I was forced to rely on for data collection is set up in such a way that it is difficult to obtain a single camera image snapshot (such as a picture of concentric circles for distortion correction). The system is configured to produce a sequence of images from a particular scan of an object. Second, the CyberWare camera has a filter installed over the lens that makes everything except for light near the laser frequency very dark, and I did not want to attempt to remove it, as I did not want to damage the equipment and its calibration. Finally, I am using an approximation for the camera field of view in the x and y directions, as well as other approximations for the geometric constants of the scanner in my data interpretation program that are likely to be much greater sources of inaccuracy when compared to errors due to distortion[3].

The second phase (triangulating the geometry) comprised the bulk of my efforts in the data interpretation part of my project. A discussion of the main ideas are included in the next section.

3.2 Theory / Mathematical Foundations

There is a large field of research literature in the area of interpreting the results of laser range-finding data. One primary source of information about various methods is given in a survey by Paul Besl [1]. Other more recent sources include papers by Curless[2] and Dorsch[3] which detail some of the limits to the accuracy of triangulation methods. The major issues I will briefly discuss below.

The initial task one faces in attempting to reason about an object's geometry from a given laser stripe contour is to find the pixel value (or interpolated pixel value) along each image scanline that corresponds to the location of the center of the laser light scattered off

[3]In addition to the other reasons given, I have already implemented a program to correct for camera lens distortion as part of the homework in this class.

of the object surface at that point. Here, we assume that the object scanned is opaque, and has diffuse reflective properties (no shiny, mirror-like surfaces that might cause erroneous results due to reflections).

One simple method of estimating the location of the center of the laser light is to find the pixel with the maximum intensity value along a given image scanline[4]. This is the method that I implemented, since I did not have an accurate enough model to justify more complicated schemes. A few of the alternatives to the maximum pixel value method include using the mean, the median, or fitting a one-dimensional Gaussian distribution to the intensity data on each scanline, and using the mean of the Gaussian as the center point. Curless[2] uses a spacetime analysis in order to achieve good results even when the object's surface reflectance varies.

One obvious question that eventually arises when contemplating the scanning process is "what happens when the object has sharp protrusions or deep recesses?" The answer of course is that these will cause occlusions of the laser stripe seen by the camera. There are a number of ways of dealing with this problem, including scanning the object from an angle which will minimize or eliminate occlusions. Others have devised methods of combining the results of several scans to produce the full object geometry[6]. In Section 4: Mesh Generation, I explain further how my software deals with occlusions.

Finally, given that we have computed the center point of the laser reflection for a particular scanline, how do we compute the corresponding 3D coordinates of the point on the object surface? Using simple trigonometry, I derived the equations necessary to accomplish this task. Figure 2 displayed on the following page shows a diagram of the scanning setup.

We define the following constants:

$$
\begin{aligned}
CAM\_X &= \text{distance from camera origin to laser origin.} \\
CAM\_Z &= \text{distance from laser origin to object origin.} \\
CAM\_\theta &= \text{angle between camera axis and object } x\text{-axis.} \\
Fov\_x &= \text{camera field of view along the } x\text{-axis.} \\
Fov\_y &= \text{camera field of view along the } y\text{-axis.} \\
MAX\_X &= \text{number of horizontal pixels in pixel plane (the image).} \\
MAX\_Y &= \text{number of vertical pixels in pixel plane.} \\
Scan\_step &= \text{distance between successive scans.} \\
Num\_scans &= \text{total number of scan images acquired.}
\end{aligned}
$$

[4] if no maximum is found, then we assume that the laser does not strike the object at that scanline. Later on, I will introduce the problem of occlusions.

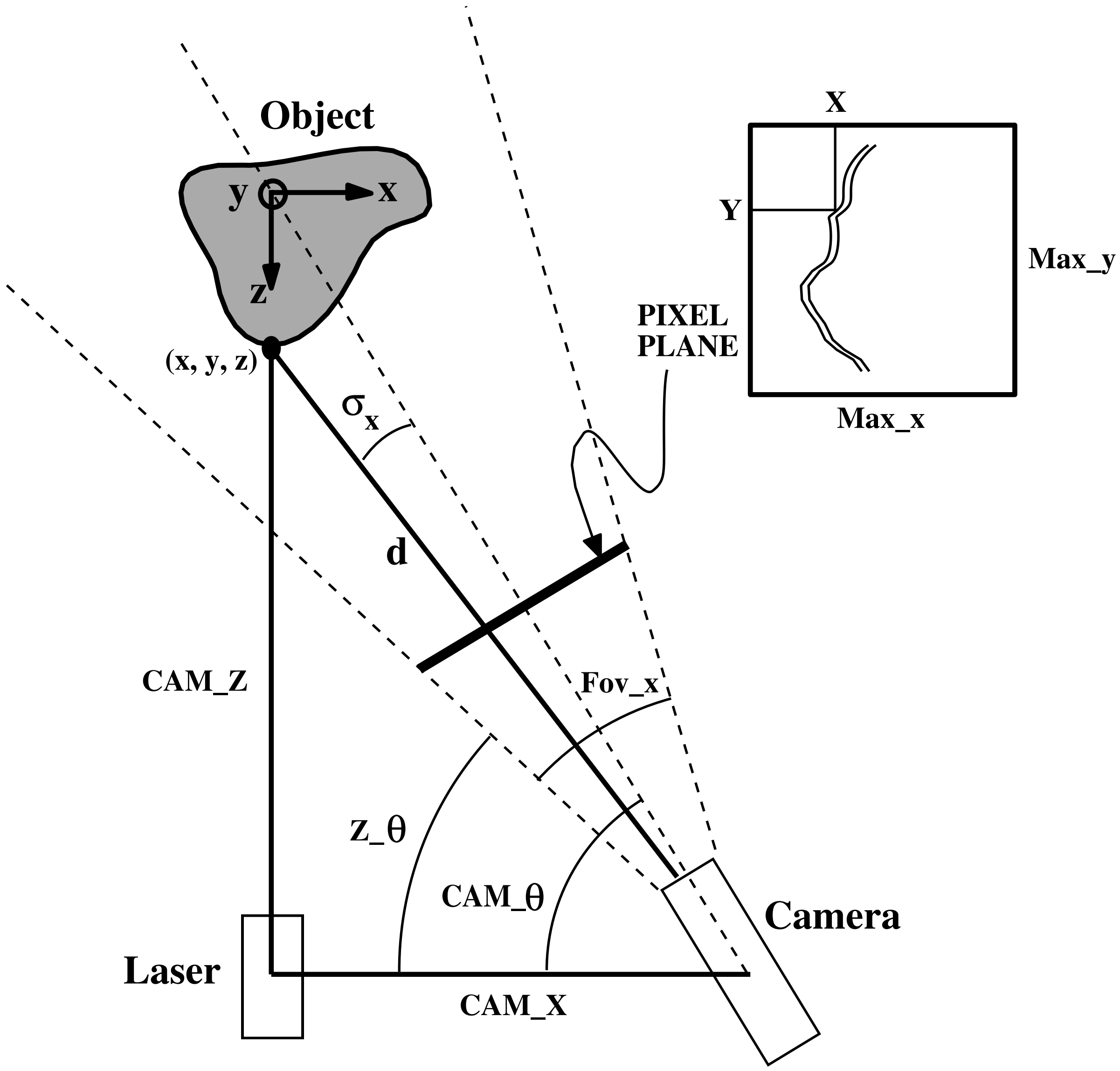

Figure 2. The geometry of the scanning setup.

We define the following variables:

$$
\begin{aligned}
(x, y, z) &= \text{world coordinate on the object surface (in object's reference frame).} \\
(X, Y) &= \text{pixel plane (image) coordinates of center of laser reflection.} \\
d &= \text{distance from camera center to point on the object surface.} \\
\sigma_x &= \text{angle between vector to point on object surface and camera axis.} \\
Z\_\theta &= \text{angle between vector to point on object surface and object } x\text{-axis} \\
f &= \text{distance from camera origin to pixel plane origin.} \\
scan &= \text{current scan number.}
\end{aligned}
$$

We are given (X, Y) and we wish to compute (x, y, z), the coordinates of the center of the reflection point on the object surface. We know x, since we are scanning along that axis, and it is constant for a given image. Picking the point in the middle of our scanning range as the origin of our x-axis, we can compute x as

$$x = (scan - (\frac{Num\_scans}{2})) * Scan\_step \tag{1}$$

Now to compute y and z, we need to define some intermediate quantities. Let d be the distance from the camera center to the point on the object surface, and be given by

$$d = \sqrt{CAM\_X^2 + (CAM\_Z - z)^2}$$

Using similar triangles, we can compute z given that we use the pixel information X, which will be small if z is large. We compute σ_x, the angle between the camera axis and the vector from the camera origin to the point on the object surface in the following manner

$$\sigma_x = \tan^{-1}(\frac{|X - (MAX\_X/2)|}{f})$$

Assuming a camera aspect ratio of 1.0, the distance to the pixel plane, f is given by

$$f = \frac{MAX\_X}{2 * \tan(Fov\_x/2)} = \frac{MAX\_Y}{2 * \tan(Fov\_y/2)}$$

Since

$$Z\_\theta = CAM\_\theta - \sigma_x$$

where

$$CAM\_\theta = \tan^{-1}(CAM\_Z/CAM\_X)$$

Now, we have all of the information necessary to compute y and z.

$$y = (d/f) * ((MAX\_Y/2) - Y) \tag{2}$$

$$z = CAM\_Z - CAM\_X * \tan(Z\_\theta) \tag{3}$$

Each point (x, y, z) is output to the file `points.dat` by the `interpret` program. If a scanline is blank, I output a point with a zero value for z, which is a signal to the `mesh` program that the data point is either occluded or does not exist for that particular scanline.

3.3 Experimental Results

The implementation of the mathematical model above is fairly straightforward, and the code is included in Appendix A. The program was run on the sample bunny data set (using 170 scan images). Below is the output of the image data interpretation, showing a large increase in the number of non-blank scanlines in the images as the bunny appears in the path of the laser. The percent useful represents the total fraction of scanlines that produced a data point on the object surface. The maximum percent useful was about %79.5 for data set number 50.

```
elaine8 8:03pm INTERPRET>interpret
SCAN  1: Number of blank scanlines = 127   percent useful = 0.0
SCAN  2: Number of blank scanlines = 126   percent useful = 0.8
SCAN  3: Number of blank scanlines = 126   percent useful = 0.8
SCAN  4: Number of blank scanlines = 127   percent useful = 0.0
SCAN  5: Number of blank scanlines = 127   percent useful = 0.0
SCAN  6: Number of blank scanlines = 127   percent useful = 0.0
SCAN  7: Number of blank scanlines = 126   percent useful = 0.8
SCAN  8: Number of blank scanlines = 125   percent useful = 1.6
SCAN  9: Number of blank scanlines = 127   percent useful = 0.0
SCAN 10: Number of blank scanlines = 126   percent useful = 0.8
SCAN 11: Number of blank scanlines = 122   percent useful = 3.9
SCAN 12: Number of blank scanlines = 114   percent useful = 10.2
SCAN 13: Number of blank scanlines = 103   percent useful = 18.9
SCAN 14: Number of blank scanlines = 96   percent useful = 24.4
SCAN 15: Number of blank scanlines = 82   percent useful = 35.4
SCAN 16: Number of blank scanlines = 71   percent useful = 44.1
SCAN 17: Number of blank scanlines = 70   percent useful = 44.9
SCAN 18: Number of blank scanlines = 66   percent useful = 48.0
SCAN 19: Number of blank scanlines = 62   percent useful = 51.2
SCAN 20: Number of blank scanlines = 59   percent useful = 53.5
SCAN 21: Number of blank scanlines = 52   percent useful = 59.1
SCAN 22: Number of blank scanlines = 51   percent useful = 59.8
SCAN 23: Number of blank scanlines = 49   percent useful = 61.4
SCAN 24: Number of blank scanlines = 48   percent useful = 62.2
SCAN 25: Number of blank scanlines = 47   percent useful = 63.0
SCAN 26: Number of blank scanlines = 46   percent useful = 63.8
SCAN 27: Number of blank scanlines = 45   percent useful = 64.6
```

...rest deleted

4 Mesh Generation

4.1 Description

The goal of this phase of the project is to create a program that inputs a set of 3D coordinates and outputs a triangular mesh corresponding to a surface that represents those points.

4.2 Theory / Mathematical Foundations

The 3D coordinates stored in the file `points.dat` are arranged in a convenient way. We know that the laser can intersect the object in only one point at each scanline in a given image, so we essentially have a depth field (or range field - hence laser range-finding). The points are output in a grid-like fashion, so the meshing program simply iterates through the points and checks to see that each vertex of a proposed triangle is "valid" (here, valid means that every vertex has a non-zero z value). Invalid triangles are discarded, whereas the rest are output to the file `mesh.dat`. Given the vertices A, B, and C in a clockwise ordering, the unit normal vector n is computed by taking the cross product of two edge vectors of the triangles

$$n = \frac{(A-B) \times (C-B)}{|(A-B) \times (C-B)|}$$

and output to the file. All of the points in the grid are iterated through in this fashion, and a set of valid triangles results. We expect that there may be some holes in the mesh, which are due to occlusions and perhaps camera noise, but in general the results I obtained were fairly clean.

4.3 Experimental Results

I ran the mesh generation program on the `points.dat` file output by `interpret.c` on the bunny data set. The output was a file of over 20,000 triangles (three vertices and a normal vector) which was stored in `mesh.dat`.

```
elaine8 8:05pm MESH>mesh
Generating mesh for 170 x 127 data point grid...
Done!  Number of triangles in mesh = 20326
```

Here is a sample of the `mesh.dat` file. The file format consists of the coordinates of each of the three vertices of the triangle specified on a separate line, followed by the components of the normal vector of the triangle.

```
-0.071000       0.014318        0.101397
-0.071000       0.012727        0.101397
-0.070000       0.012713        0.103633
-0.000004  0.000000  0.000002
-0.064000       0.077954        0.101397
-0.063000       0.076448        0.099155
-0.063000       0.077867        0.103633
-0.000004  -0.000004  0.000001
-0.060000       0.033446        0.099155
-0.060000       0.031818        0.101397
-0.059000       0.031782        0.103633
-0.000004  0.000002  0.000002
-0.058000       -0.048716       0.060230
-0.057000       -0.050696       0.046108
-0.057000       -0.048659       0.062564
-0.000004  -0.000016  0.000002
```

...rest deleted

Now, all we need to do is visualize our data!

5 Visualization

5.1 Description

The goal of this phase of the project was to facilitate an easy and efficient method of visualizing the surfaces computed by the meshing program. My primary goal here was to produce an executable for the SGI platform that would allow a user to interactively view the triangular mesh of the original scanned object.

5.2 Experimental Results

To aid in the construction of a graphical user interface, I employed the public-domain Forms 2.2 FORMS library and windowing toolkit. Appendix C includes all of the source code and forms definitions files. The program reads the triangular mesh data from the file `mesh.dat` and displays a shaded, orthographic view of the mesh in the main window. The auxiliary window contains buttons and sliders to change the point of view interactively. Figure 3 shows an SGI screen snapshot of the program while executing after having loaded the bunny mesh data. Figure 4 shows a detail shot of the bunny visualized.

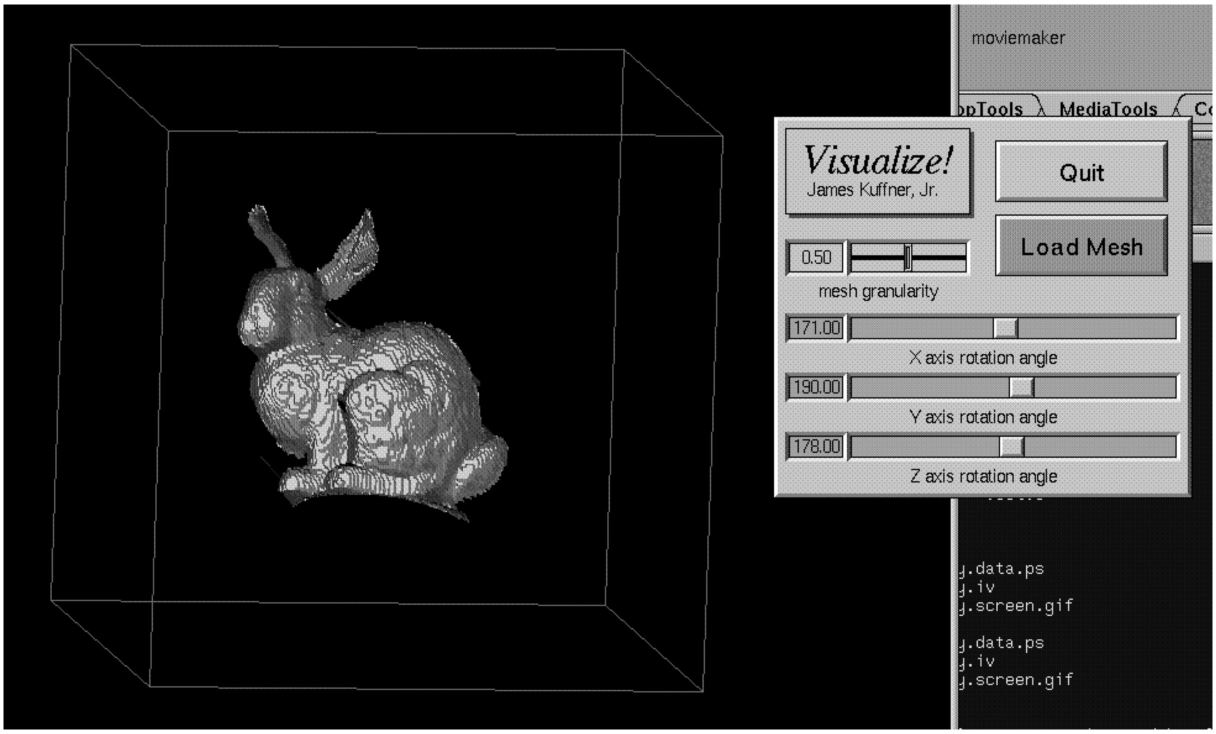

Figure 3. An SGI screen snapshot of the mesh visualization program. The bunny mesh shown here contains over 20,000 triangles.

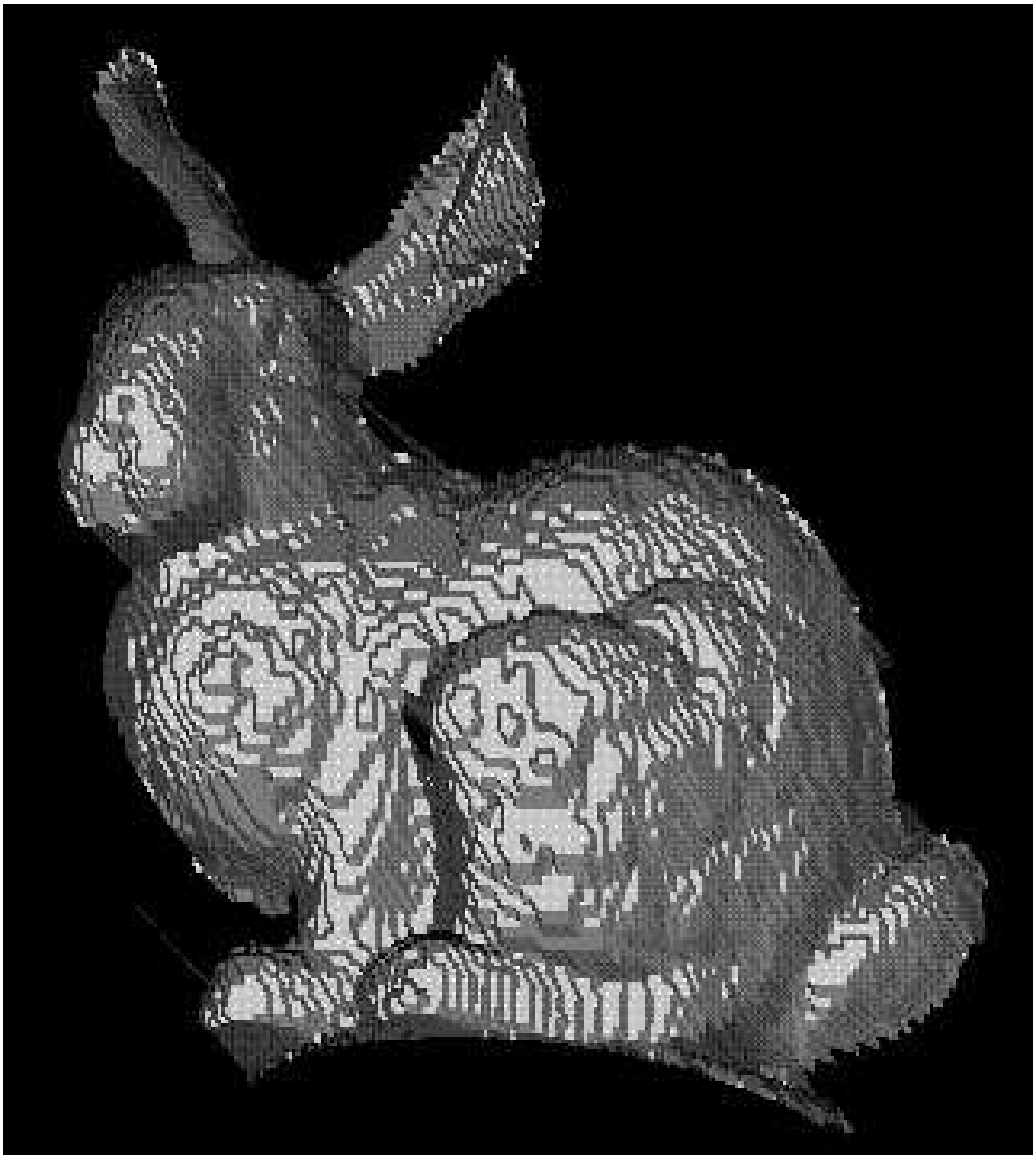

Figure 4. A closeup of the bunny surface mesh. Even some of the minute detail on the bunny model was captured by the laser triangulation.

6 Conclusion

As we have seen in class this quarter, acquiring shape and appearance essentially requires solving an inverse rendering problem: given a set of images, solve for scene illumination, sensor geometry, object geometry, and object reflectance. Laser range-finding is one method of reconstructing information about one aspect of this task: recovering object geometry. One can imagine a more powerful version of this technique that reconstructs the color information of the object as well as simply the geometry (perhaps by using another camera). There are several major drawbacks to laser range-finding, notably the fact that the object must be static and fairly close the the robot, and that the technique generally fails outdoors in the presence of intense sunlight. However, as a solid method for recovering object geometry, and as a testbed technique for learning about computer vision and robot navigation systems, I believe that it is an extremely useful method, as well as being fun to implement.

References

[1] P. Besl, "Advances in Machine Vision," chapter 1 - Active optical range imaging sensors, pp. 1-63. *Springer-Verlag*, 1989.

[2] B. Curless and M. Levoy, "Better optical triangulation through spacetime analysis," *to appear in ICCV '95*, 1995.

[3] R.G. Dorsch, G. Haulser, and J.M. Herrman. "Laser triangulation: fundamental uncertainty in distance measurement." *Applied Optics*, 33(7):1306-1314, March 1994.

[4] J.L. Mundy and G.B. Porter. "Three-dimensional machine vision", chapter 1 - A three-dimensional sensor based on structured light, pages 3-61. *Kluver Academic Publishers*, 1987.

[5] T. Strand. "Optical three dimensional sensing", *Optical Engineering*, 24(1): 33-40, Jan-Feb 1983.

[6] G. Turk and M. Levoy. "Zippered polygon meshes from range images". In *Siggraph 94 Conference Proceedings*, pp. 311-318, July 1994.

7 Appendices

Appendix A	:	interpret.c (data interpretation)
Appendix B	:	mesh.c (code to generate a triangular mesh)
Appendix C	:	visualization (code to visualize the mesh on an SGI workstation)