Dense 3D Reconstucion
COMS 6733 3D Photography Final Project

Austin Reiter and Hao Dang

May 6, 2010

1 Motivation

Current 3D reconstruction methods using stereo vision can produce good results, but they tend to yield very sparse outputs and suffer from several challenges. One of them is the well-known correspondence problem, whereby similar features in different images are often too ambiguous to match reliably. The problem is accentuated when the object being scanned is textureless. Under these circumstances, methods that are widely used for correspondence determination, such as SIFT, normalized cross-correlation of corner features, and histogram descriptor approaches, will not be able to produce reliable results for matching stereo features for 3D triangulation.

Our project consisted of two different approaches, each trying to alleviate these issues. One is based on photometric stereo. The second is based on a two-camera stereo system with a finely-controlled laser pointer. The former did not yield the expected results, but it supplied us with useful information to succeed with our second approach, which turned out to be much more successful. Both methods tried to take advantage of the Staubli robot arm for its ability to achieve fine-tuned positional accuracy. In the first case, we used it to position the light source, and in the second case we used it to control the motion of the laser pointer. In this report, we will discuss the procedures for both approaches, analyze the influential factors for the failure of the first approach, and show the experimental results of the second approach and discuss some areas for improvement.

2 Photometric Stereo

The most attractive attribute of photometric stereo is that it frees us from pixel correspondences, and thus allows us to reconstruct at the resolution of the camera. However, this method is normally used to recover surface normals on top of 3D data that has already been collected. We aimed to constrain the problem further to attempt to recover the 3D geometry (i.e., point clouds) *as well as* surface normals for photorealistic reconstructions. We used a high dynamic range camera for maximal sensitivity to changing lighting conditions at each surface point. The Point Grey Dragonfly2 camera sufficiently supplies 12-bits per pixel at a resolution of 1024x768, and so detecting how lighting changes for each light location is sufficiently captured.

2.1 Theory

The theory of photometric stereo is based on a very important assumption of the object surface, called Lambertian Reflectance. This states that rays from a light source affects all parts of a surface equally, regardless of viewing angle. Based on this assumption, we can formulate the image intensity at a pixel p as follows:

$$I_p = \rho L \cdot n_p \tag{1}$$

where ρ is the surface albedo, I is the image intensity at pixel p, L is the lighting direction, and n_p is the surface normal of the object at pixel p. If we continue to consider the light source as a point light, for L we have:

$$L = \frac{Loc_p - Loc_{light}}{||Loc_p - Loc_{light}||} \tag{2}$$

where Loc_p is the 3D location of the surface point corresponding to pixel p and Loc_{light} is the 3D location of the point light source.

In Eqn. 1 and 2, we note the following:

1. I_p is known from the pixel intensities collected from the camera
2. Loc_{light} is known from the robot arm position. This required a calibration of the robot to the camera coordinate system.
3. The surface point 3D location Loc_p and the corresponding surface normal n_p (as well as surface albedo) are unknown

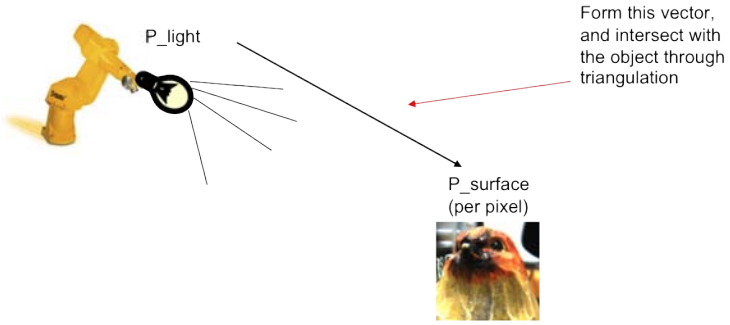


Figure 1: A visual setup of the geometry proposed in our photometric theoretical idea to recover both 3D points and surface normals from changing light sources of a static scene.

4. If we take multiple images sequentially, ρ, Loc_p and n_p will not change

Considering the constraints above, we proposed to be able to calculate Loc_p and n_p by taking multiple images to form a system of equations and solving it.

To constrain the problem further, we considered the imaging geometry setup between the light source and the camera, and the goal was to triangulate the light ray from the light source, through the object 3D location, with the pixel ray from the camera through that same object point location. Simply speaking, a 3D point in the space L_p when re-projected onto the camera should agree with its camera pixel location p. If the pixel re-projection does not agree with the original camera pixel location, chances are we obtained a bad reconstruction result. Figure 1 shows the visual interpretation of this proposed idea.

2.2 Reconstruction Attempts

Depending on different groupings of unknown variables, we tried two ways for the reconstruction based on the theory we developed in Sec. 2.1. Note that in the following, where we take several images of the same object under different lighting conditions, when cycling through the image pixels we always choose the top 3 (brightest) measurements per pixel to solve the system, as suggested in the original photometric stereo description. In taking 9 images per object, we get robustness to different lighting angles as the shape of the object is more or less occluded.

2.2.1 One-Phase Reconstruction

This method worked somewhat as a *black box*. The entire process is described below:

1. We kept the camera at the same position during the whole process.

2. In the environment, there is only one point light source that emits light, per image (i.e., all ambient lighting shut off).

3. The location of the point light source is obtained in a common coordinate system through some calibration device, e.g. Microscribe 3D digitizer.

4. Using the above setup, multiple images were taken when the light source was placed at different locations, via the robot arm. In our experiments, we took 9 images, each with the light source at a different location.

5. Using these images, we fed them to Eqn. 1 and 2 to form up a big non-linear system of equations, **per-pixel**. We used MATLAB's optimization toolbox to try and solve this system of equations to recover all unknown parameters at each pixel separately.

2.2.2 Two-Phase Reconstruction

By examining the equations more closely, we thought that it might be possible to first calculate the normals at each pixel by using a parallel lighting setup, using the traditional photometric stereo approach (i.e., first solving a simple linear system), and then using that to compute the 3D locations of each point using point light source. The entire process differs from the One-Phase Reconstruction in the following aspects:

1. We build a parallel light source by putting the light bulb far away from the object.

2. Take several images by changing the lighting direction, again with the robot arm.

3. Using this experimental data, we fed them into Eqn. 1 to solve for n_p and ρ.

4. We switch from parallel lighting to point source lighting.

5. Multiple images were taken as in One-Phase Reconstruction.

6. Using these images, we fed them into Eqn. 2. Now the system of equations is much simpler, with fewer degrees of freedom than before because the surface normals and surface albedos for each surface point at each pixel is already obtained. MATLAB was again used to solve for the locations of the surface points.

2.3 Conclusions

We did not get reasonable reconstructions from these methods. The possible factors that probably made this approach fail may include:

- The Lambertian assumption for the reflectance properties of the object is not necessarily true for most real-world objects. This could be fixed by applying some paint that better approximates the Lambertian property, or by estimating the true BRDF of the object being scanned.
- The point light source is difficult to simulate in our experiment environment. This problem could be relaxed with a more accurate lighting system.

In addition, it was hard to even view if our surface normals were correct or not (when obtained separately) because we didn't have good 3D data to plot on top of. We could view them in the 2D imaging plane, which helped somewhat, but overall this indicated that we needed a new method.

3 Two-Camera Stereo

Next we describe the subsequent method using a 2-camera stereo system. In order to solve the pixel correspondence problem, we used a laser point light source which can be easily detected and located by thresholding the image intensity. A Staubli Robot Arm is used to control the light source to obtain fine pixel reconstruction density. See figure 2 for an image of our hardware setup. We descibe the setup in more detail below.

Figure 2: The hardware setup of the two-camera stereo system with a laser pointer attached to a robot arm, as it scans the bird.

3.1 Experimental Setup

The hardware system consisted of a laser pointer, a Staubli robot arm that controlled the laser pointer with fine movements, and three cameras, two gray scale Prosilica gigabit ethernet cameras with 25mm lenses for camera triangulation, and one color camera for texturing. The color camera, called the *texture camera*, is only necessary because the stereo cameras only support gray-scale color formats. Because we wanted photorealistic 3D reconstructions, we used an additional color camera for the texture mapping step (described in more detail below). Our color camera was a Point Grey Dragonfly2 camera running in 1024x768 RGB color mode, and we used it to snap single shots in the beginning of each scan.

Although many methods use only a single camera with a laser pointer, we chose a stereo system to simplify the calibration process. Using a checkerboard pattern, we are able to simultaneously obtain the extrinsic transforms between all cameras in a single shot. Then we could move the laser pointer however we wish without worrying about it's position and orientation. Thus, the laser pointer is simply a means to easily extract stereo correspondences in the 2 reconstruction cameras. To ease the problem further, we dimmed the lights so that the brightest part of each image was only the laser pointer, and a gray-scale thresholding could pick out the laser point blob.

A note on the laser point in the images: because our cameras were very high resolution (1620x1220 pixels), the laser pointer in each image was more of a *blob* rather than a point. This presents a problem because each image assumes a laser point, and so we simply took the centroid of the blob in each image. This may not be exactly correct, and future methods could try to discern the correct laser image position more precisely, or alternatively try to match the entire blobs using shape matching methods, which would also produce more points per image. In our case, we wish to simplify the problem as much as possible and just take the blob centers as our point-of-interest. Thus, each image frame provides a single 3D point on the object being scanned, and we move the laser pointer to collect all surface points that are visible to both cameras.

The laser pointer is attached to the Staubli arm. In order to take advantage of the fine-tuned motions of the robot arm, we manually positioned the arm to a starting point, and wrote a C++ program to scan the object in a planar motion from the starting position and orientation. We set a pre-defined speed and scan-line spatial step in order to obtain as dense a reconstruction as possible. Limitations to this setup were due to the 15fps video capture speed of the cameras as well as the processing time of each triangulation, and so in the end the reconstruction system processed at about 12Hz. Note that this is a limitation of our particular setup, but with a high-speed, high-resolution camera system, the process could be sped up significantly.

3.2 Software Utilities

Three main software utilities are running at the same time while the scan is in progress. It is worthwhile to note that multi-scan registration and fusion is done as an independent step, and this is described in more detail later on.

Robot Controller The robot controller controls the end effector of the Staubli arm that holds the laser pointer. It moves within the plane that is orthogonal to the end effector's z direction. It moves along pre-defined lines within a rectangular region. Since the query frequency of the cameras is the limiting factor, the speed of the laser point determines the fineness of the scanning. Generally, the slower it moves, the denser the point cloud will be and thus the more accurate the model.

Point Cloud Generator A separate program is run on another computer that connects with the three cameras. It queries the two gray scale

cameras while the laser pointer is moving on the object. The images obtained from the two cameras are then thresholded to detect the laser point location, and triangulation is performed to obtain the 3D location of the corresponding pixels. At the same time, the 3D location is back-projected to the texture camera to get the correspondence in the texture image.

Online Viewer An online viewer is run on a third computer which monitors the real-time progress of the scan. Whenever the cloud generator program gets a new 3D point, it sends to the viewer, via UDP, the point location, the pixel RGB value, and the corresponding pixel location in the texture image obtained from the color camera. At the same time, the viewer program keeps a buffer of all 3D points collected so far, and intermittently performs a 3D Delauney Triangulation to approximate the surface mesh online. As more points are collected, the surface mesh becomes more detailed. Finally, each triangle is texture-mapped using the texture image, to produce photorealistic imagery in the reconstruction scene. See figure 3 for a sample screenshot of our online viewer.

In order to ease our experiments, we added logging capabilities to the point cloud generator so it could write out all UDP messages to a text file every time it was sent over the socket. This text ASCII file could then be

Figure 3: A sample screenshot of our online viewer, as it shows an online reconstruction of the bird.

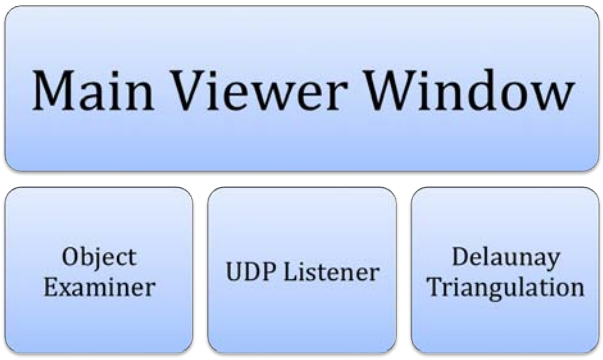


Figure 4: The workflow of the multi-threading structure of the real-time scanning viewer.

loaded by the viewer, for offline purposes, to view the reconstruction. This allowed us to post-process some data, in the case of bad points that ruined the overall appearance of the mesh triangulations. Using simple point-to-point distance thresholds of points that were gather consecutively, we are able to determine which points are incorrect and ignore them so that the mesh is more accurate.

3.2.1 Multi-threaded Viewer

A serialized version of the viewer did not work well. When the viewer was trapped in a triangulation procedure, it simply stopped listening to the UDP port and resulted in missing a lot of the raw points. This proved to be fatal to the reconstruction process. Another defect of the serialized viewer was caused by the UDP listener which constantly queries for points, and thus consumed a lot of CPU work time. This made the viewer not promptly responsive during user interactions.

To solve these two problems we re-designed the architecture of the viewer and made it multi-threaded. The main window of the viewer is the parent thread. The UDP listener is one child thread, and the routine for mesh triangulation is another child thread. Figure 4 shows the workflow of this multi-threading structure.

3.3 Camera Triangulation

As each pair of images was collected from the stereo Prosilica cameras, we threshold both images with a pre-defined gray-scale threshold to produce a *blob* in each image. For each blob, we compute the mean pixel location, and this produces an effective **stereo match**. This match is fed into a stereo triangulation procedure, which solves for the best 3D location that serves as an intersection of the pixel rays emanating from each camera at these pixel locations, in the least-squares sense. To reduce bad intersections, we take the resulting 3D location, and back-project into both cameras, and compute the pixel errors. If the pixel error back-projections exceeded a pre-defined threshold, the 3D point was thrown away. Then, because all 3 cameras are calibrated to each other, we can back-project from this 3D location into the texture camera to get the corresponding 2D pixel location which will be used for texture mapping, next.

3.4 Texture Mapping

As each 3D point (and corresponding 2D texture location) is collected by the UDP thread listener, a buffer is filled. We set a pre-defined frequency which tells the Delauney triangulation thread to re-triangulate what's currently in the point buffer to provide us with a new approximation to the surface mesh. All of this is done so as to not interfere with each other as well as the main window, so the user can manually inspect the object (i.e., rotate and translate with the mouse) without missing measurements. Once the newest triangles are created, for each triangle, we take the 3D vertices of the triangle, and cut-out the corresponding 2D image patch from the texture image. Then, OpenGL texture maps that image information onto that 3D triangle, and all triangles are combined to form the full reconstruction of the object, online.

3.5 Multi-Scan Registration and Fusion

In order to have a complete model of the object, we need to be able to combine multiple scans of the object by showing the object at different angles to the cameras. Along with this comes the problem of how to find the transformations between the sequential scans and merging them into a complete model. In our work, we used a well-known algorithm, Iterative Closest Point (ICP), to calculate the rigid transformations from two scans.

To prove the method, we take 2-3 scans of the same object but from different angles. The point clouds must have some overlap. The basic steps to find the transformation between them can be described as follows:

1. Manually select corresponding points in two scans
2. Feed these points to ICP for the calculation of the rigid transformation
3. Apply the transformation to the original point set which contains not only the overlapped points but also those not overlapped. This step merges the two full scans into the same coordinate system

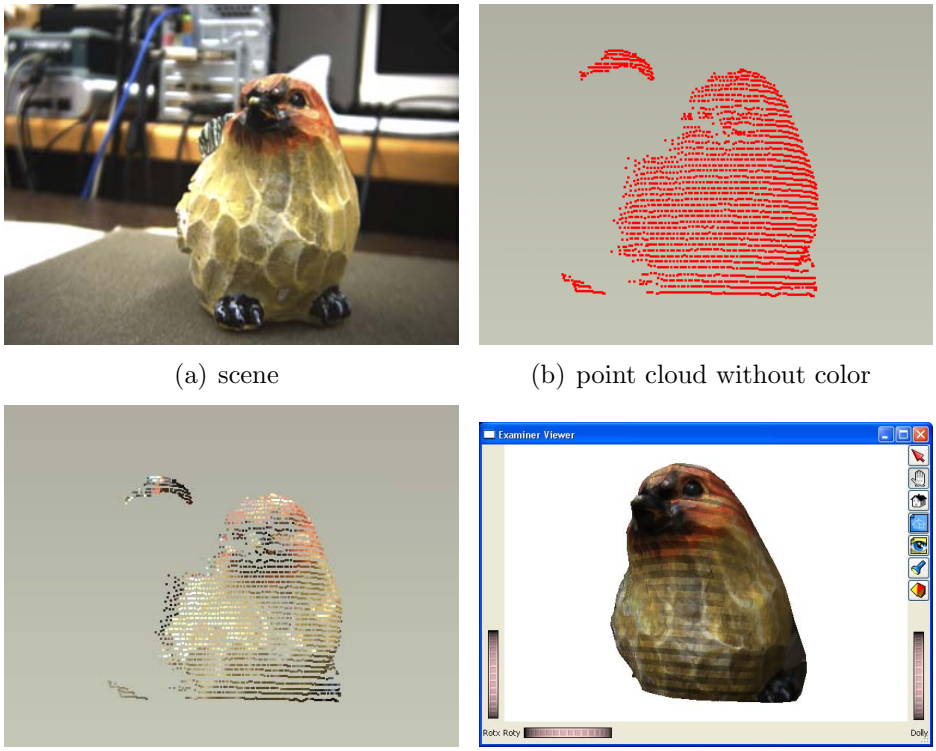
Texture mapping, although possible, is not straightforward when combining multiple scans. The reason is that the texture image is retrieved by back-projecting from a 3D point in a calibrated camera's coordinate system to obtain the 2D pixel locations. However, the ICP algorithm changes the coordinate system of some of the 3D points, and so this must be accounted for when back-projecting into the proper texture image (i.e., there are now several texture images).

In addition, for areas of overlap there's the issue of which view is the *best* view to use for the texture mapping. For example, you probably want the most straight-on view as possible, but this can only be obtained by know some information of the surface normal and comparing that to the viewing angle of the camera with respect to each surface normal. In theory, you want a camera who's viewing angle is closest to the surface normal. If the viewing angle is too close to orthogonal to any surface normal, the view from the camera is probably a bad view. For these reasons, we only propose the method for merging multiple scans using this system and cite possible ways to texture map later on.

3.6 Results

3.6.1 Single Scans

We did scans on several objects, including a bird statue (Fig. 5), a sumo wrestler (Fig. 6), a frog(Fig. 7), and a mouse (Fig. 8). There are artifacts in the surface mesh that are visible as scan lines; these are the spatial *jumps* that the laser pointer makes from one line to the next in its planar scan. The smaller we make that jump, the less the artifacts will be present, at the cost of a much longer scan.



(a) scene (b) point cloud without color

(c) point cloud with color per point (d) mesh with texture

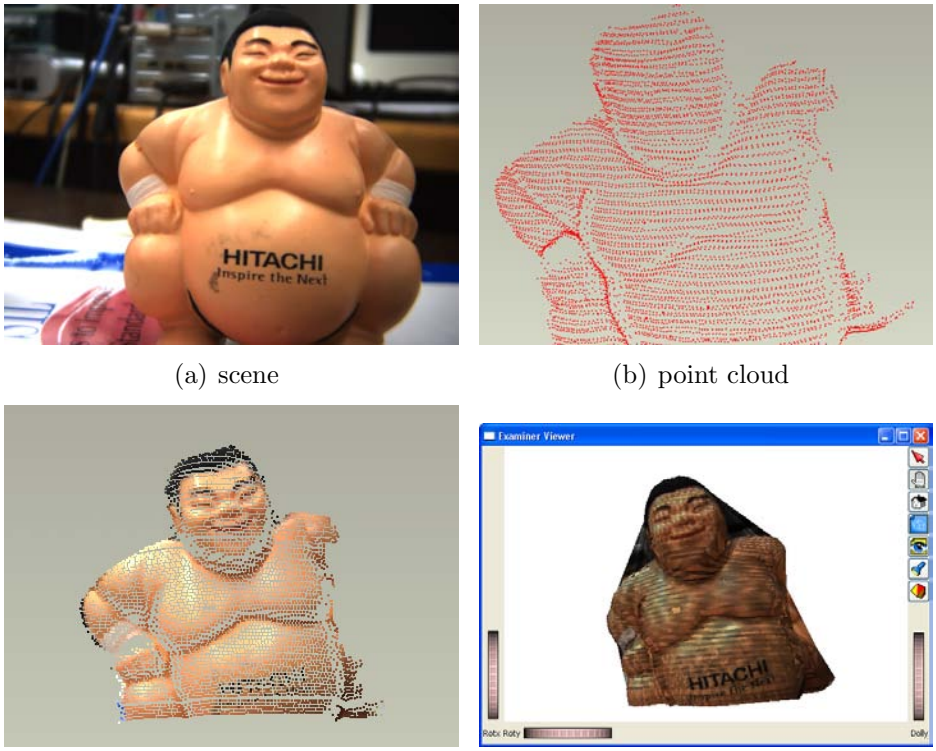
Figure 5: The result of scanning a bird statue.

In the following, we show each experiment including the scanning scene, the point cloud without color, the point cloud with each point colored by the image pixel color, and the triangulated mesh with texture mapping.

3.6.2 Multi-scan Fusion

One experiment was done for multi-scan registration and fusion. Based on the two point clouds, we manually selected the corresponding points and applied the ICP algorithm to them. Based on the results of the ICP algorithm, we transformed those two point clouds into the same coordinate system. Fig. 9 shows the result.

Ways this process could be improved would be to automate the correspondence selection procedure. Using the texture images, it would be possible to run a standard feature matching algorithm (i.e., SIFT) and use these to

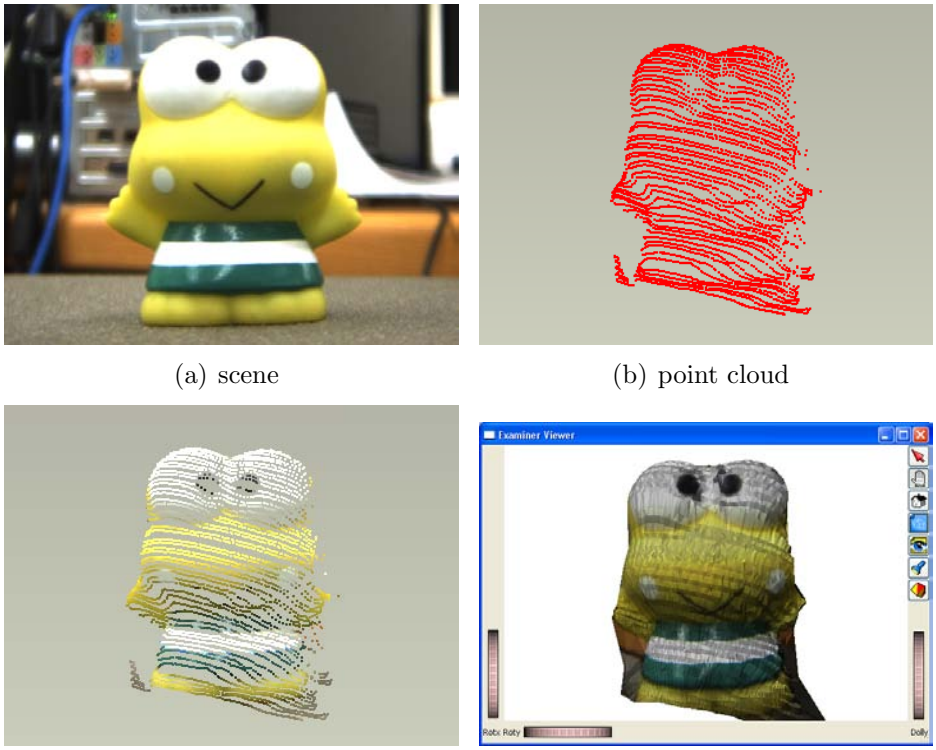


(a) scene (b) point cloud

(c) point cloud with color per point (d) mesh with texture

Figure 6: The result of scanning the sumo wrestler.

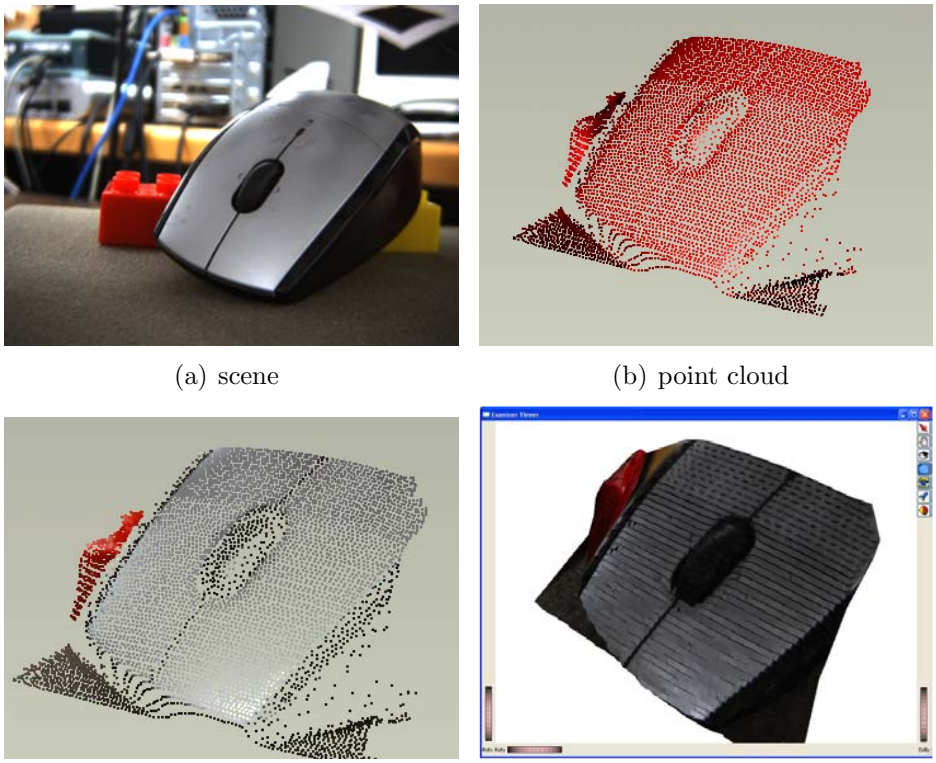
look-up in the list of 3D-2D correspondences, those that were computed by the stereo system and then used for texturing the surface mesh. If we find a valid SIFT match for which we also have a 3D point reconstructed, we could use this in a RANSAC-like approach to computing the best transform through ICP. Although we didn't have time to explore this idea, our approach certainly leaves this open as an option to further improve this system.



(a) scene (b) point cloud

(c) point cloud with color per point (d) mesh with texture

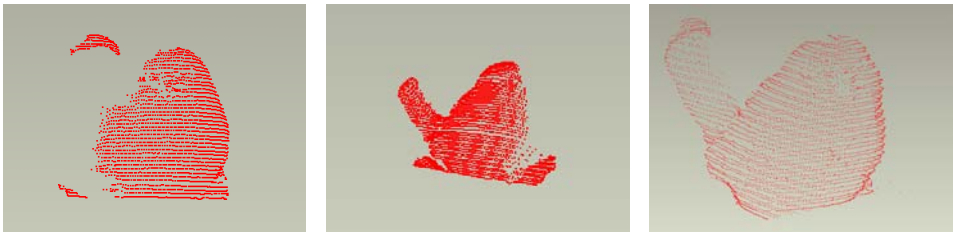
Figure 7: The result of scanning a frog statue.



(a) scene (b) point cloud

(c) point cloud with color per point (d) mesh with texture

Figure 8: The result of scanning a computer mouse.



(a) scan 1 (b) scan 2 (c) fusion of scans 1 and 2

Figure 9: The result of the multi-scan registration and fusion.