6.838: Advanced Topics in Computer Graphics: Computational Fabrication, Spring 2013

Assignment 1: 3D Photography and Range Processing Pipeline

Figure 1: A real object and a corresponding 3D model.

The code and the report are due on February 21st, 11:59pm

In this assignment, the goal is to build a simple 3D scanner that employs a projector-camera (procam) setup. You will also use a range processing pipeline in order to obtain a complete 3D model of a real object.

First, you are going to familiarize yourself with the hardware and the corresponding image acquisition and projection software. Next, you are going to geometrically calibrate the projector-camera system. Then, you are going to write the code to project and capture required images and to construct a 3D point cloud. You are going to acquire 3D point clouds from a set of views for a given real object. Finally, you are going to register all the scans and perform surface reconstruction of the object.

This first assignment does not require writing a lot of code. However, you will find it challenging if you have not worked with a projector-camera system before. The remainder of this document is organized as follows:

1. Installing Required Software
2. Hardware
3. Capturing Calibration Images

4. Capturing Images for 3D Scanning
5. Geometric Calibration
6. Computing Range Images
7. Computing Range Surfaces
8. 3D Registration of Range Surfaces and Surface Reconstruction
9. Hints
10. Report
11. Optional Features

1 Installing Required Software

The following software is required for development under (K)Ubuntu Linux. We strongly recommend using Ubuntu and the supplied software. There are no additional credits awarded for developing your own software.

1. **GUVCView**
   GUVCView is used to control the camera parameters. Go to Ubuntu software center and search for **GUVCView**. You can also install it from the commandline using

   ```
   sudo apt-get install guvcview
   ```

2. **Matlab**
   If you have a CSAIL certificate, download it from `http://tig.csail.mit.edu/wiki/TIG/HowDoIInstallMatlab` (use option 1). If you do not have a CSAIL certificate, download it from `https://matlab.mit.edu/`.

3. **Camera Calibration Toolbox** (requires Matlab)
   Download it from `http://www.vision.caltech.edu/bouguetj/calib_doc/download/index.html`.

4. **Camera Projector Calibration** (requires Matlab)
   Download it from `http://code.google.com/p/procamcalib/downloads/list`.

5. **OpenCV**
   OpenCV can be used to capture images from the camera. Alternatively, you can hack the GUVCView and if you work on Windows you can use DirectShow. DirectShow provides an example called "amcap". Our starter code is developed using OpenCV 2.4.0 (`http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.0/`).

You will need gcc and cmake to build OpenCV. In linux, you will also need libv4l-dev(video for linux). It should be slightly easier to set up OpenCV on Windows or Mac.

In order to install these libraries in Ubuntu you can use:

```
sudo apt-get install build-essential
sudo apt-get install cmake
sudo apt-get install libv4l-dev
```

Once you have installed these libraries, *cd* to the directory where you extracted OpenCV archive and type:

```
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE ..
```

Verify that the output contains this line:

```
V4L/V4L2:                      Using libv4l
```

Then proceed to:

```
make
sudo make install
```

Note that, you can uninstall OpenCV using:

```
sudo make uninstall
```

If cmake throws an error complaining about a missing package, you can install that package using *apt-get*. If you are not sure what dependencies to install, you can refer to: `https://github.com/jayrambhia/Install-OpenCV/tree/master/Ubuntu/2.4` However, note that you do not need all the packages listed in the scripts. Only libv4l-dev is essential.

To verify that OpenCV is configured properly for image capture, you can run an example app: `http://opencv.willowgarage.com/wiki/CameraCapture`.

6. **OpenGL** and **GLUT**
   These are used for displaying simple patterns on a projector. Note that there are many alternative software solutions for this task – any GUI API should work.

   To install OpenGL and GLUT:

   ```
   sudo apt-get install libgl1-mesa-dev
   sudo apt-get install libglu1-mesa-dev
   sudo apt-get install freeglut3-deva
   ```

7. **Our starter code**
   We provide a starter code for image capture and display. To compile the code, go to the directory containing the C++ code and type:

   ```
   cmake .
   make
   ```

   The script "run.sh" takes in an integer argument i, creates a directory "data$i$" and saves captured images into the directory. You can modify this code or you can write your own display-capture app.

8. **MeshLab**
   This is a software package that you will use for geometry processing. You can download it from `http://meshlab.sourceforge.net/`.

2 Hardware

You will use a pocket projector(3M® MP160 Pocket Projector) and a camera (Logitech® Webcam Pro 9000). You can also use a turntable and a piece of black cloth. You can set the camera resolution to 1600×1200 and the projector resolution to 832×624. You can experiment with different resolutions; however, make sure that you change *capture.cpp* accordingly. The camera-projector are rigidly mounted using an acrylic plate.

3 Capturing Calibration Images

1. Plan ahead. You should capture calibration images and then scan the objects immediately after that. In this way you can avoid touching or bumping the camera-projector mount and thus invalidating the calibration.

2. Find a large rigid planar object (e.g., a piece of cardboard).

3. Print a calibration pattern (provided in the starter code folder) and attach it to the plane in a corner. Measure the grid size of the pattern.

4. Find a room that can be turned dark.

5. Set up the black cloth and the turntable (optional) as shown in Figure 2.

6. Adjust projector focus. Estimate a desired distance for scanning. Adjust the projector focus so that the projected image is sharp at the distance. The knob is at the front of the projector.

7. Display the grid pattern on the projector, take a screenshot from the projector and save the corresponding image (use bmp format).

Figure 2: The pro-cam system and an object.

8. Start GUVCView. Uncheck Auto Focus. Manually change focus to desired distance. Capture about 10-15 images while holding the cardboard at different angles and distances from the camera. Make sure that both the printed and projected grid patterns are visible. Figure 3 shows a few examples.

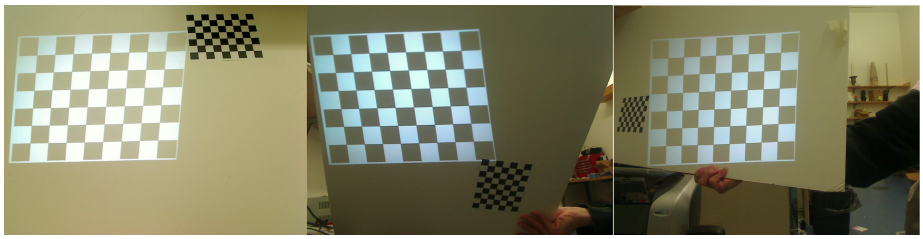

Figure 3: Examples of calibration images.

4 Capturing Images for 3D Scanning

1. Use GUVCView to change the camera settings. Uncheck White Balance and Auto Focus, set Exposure to manual. Tune the exposure so that the scanline on your object is visible and everything else is dark. Tune the parameters until you get a clean scanline similar to Figure 4.

2. Close GUVCView.

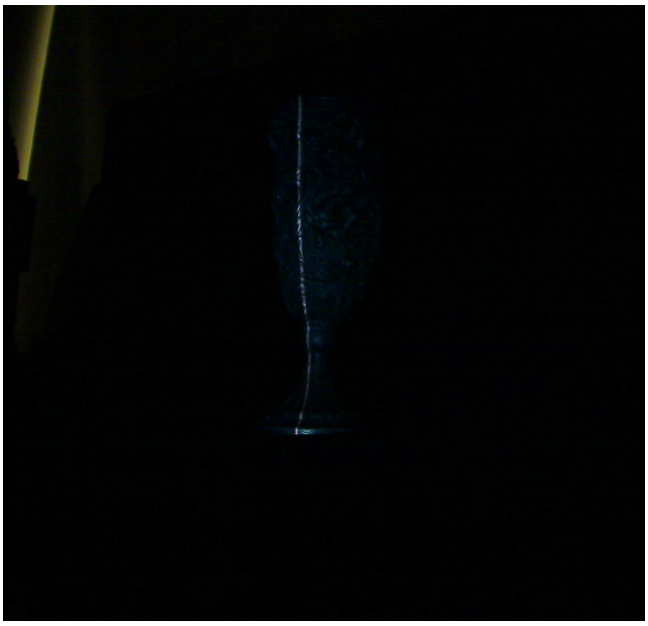

Figure 4: An example scanning image.

3. Complete and run the provided C++ starter code. You can safely skip images in which the scanline is not on your object to shorten the scan time.

5 Geometric Calibration

First, you are going to geometrically calibrate the camera.

1. Make a backup copy of your calibration images just in case they are overwritten accidentally.
2. Convert the images to gray-scale *tif* format using, for example,

   ```
   mogrify -colorspace gray -format tif *.png
   ```

3. Launch Matlab. Add camera calibration toolbox and procamlib to your Matlab path.

   Right click on the folders→Add to Path→Selected Folders.

4. Go to the directory that contains your calibration images. Use *cd* or double click on the folder.
5. Type:

`calib_gui`

in Matlab.

6. Click Standard.

7. Click Image names.

   Enter image name prefix such as "Image-". Enter "t" for image format.

   A figure showing all your images should pop up such as in Figure 5. (You can close the figure).

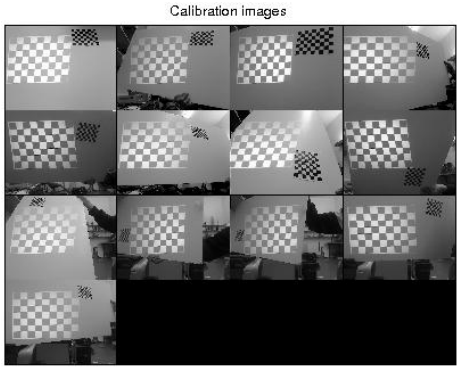

Figure 5: List of all calibration images.

8. Click extract grid corners and follow the instructions. wintx and winty are window sizes for the corner search. Setting it between 7 to 12 usually works. Hit *Enter* to use automatic grid counting.

9. Click on the grid corners starting from the upper left corner. You do not need to be extremely precise for this task. You can maximize the figure to increase your accuracy.

   Once you identify four corners, your result should be similar to Figure 6.

   Now, you have to enter the width and height of grid cells using the Matlab command window. Enter your measured values (they do not have to be extremely accurate). The cell size is $15mm \times 15mm$ if printed on US letter paper. The rest of the grid points will be automatically extracted as shown in Figure 7. When prompted "Need of an initial guess for distortion?", hit *Enter*. You do not need to provide a guess for distortion parameters.

10. In case, the image feature extraction does not work for some images, you should rerun the extraction step for these images. (After extraction for all images, you can fix those images by clicking corner extraction again and provide the image numbers you want to fix.)

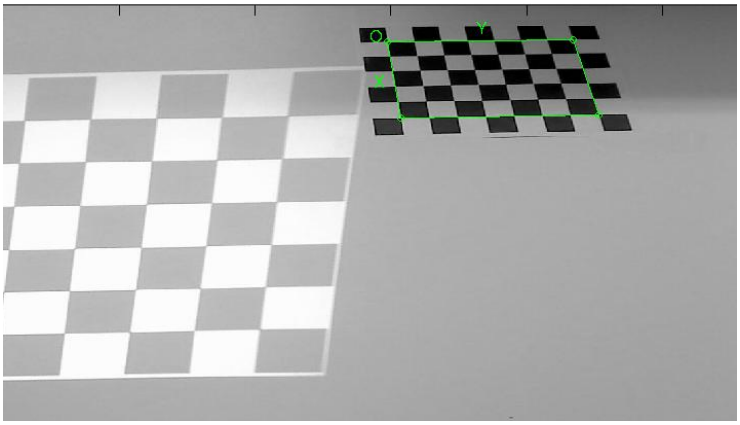

Figure 6: Extracting corners for the first image.

11. Click calibration. You can optionally let the toolbox re-extract corners automatically and then run another calibration. You should have a calibration error around 0.2.

12. Click save and then exit.

Now, the camera is calibrated. You can proceed to calibrating the camera-projector pair.

1. Type:

   ```
   cam_proj_gui
   ```

2. Click "load camera calibration". Click "set projector's calib. images" and hit *Enter* to use the same images as the camera calibration.

3. Click "Ray-plane intersection". Click grid corners of the projected image as shown in Figure 8. Extract corners for all images as well as the screenshot of the projector image you saved in Section 3.

4. Click "Calibrate the Projector".

5. Click "save".

6. Identify the relevant parameters for the camera and projector calibration. The parameters are used in *ext_calib_cam_proj.m*. *BASEk* contains the transformation matrix of the projector. *IPk* contains the projection matrix of the projector. Similarly, *BASE_cam* and *IP_cam* contain the camera calibration.

   The coordinate system puts the camera at the origin with its x-axis pointing right, y-axis pointing down and z-axis pointing front.

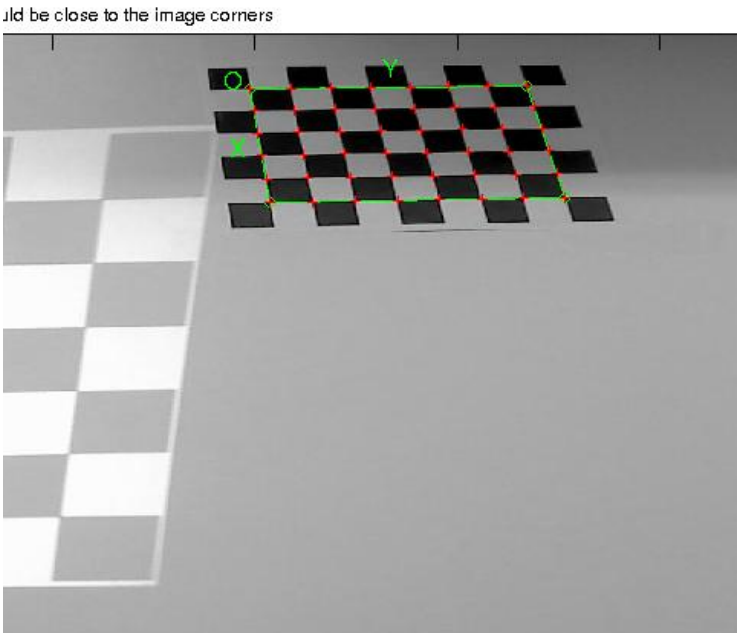

Figure 7: Automatically extracted grid points.

6 Computing Range Images

The goal of this step is to use the acquired light stripe images of the calibrated projector-camera system in order to compute a range image. You will need to implement this step (you can do it in C++ or in Matlab). We describe an outline of the approach.

Each image pixel corresponds to a camera ray. For each pixel, determine the frame with the maximum value. You can plot the radiance value for a pixel for all frames to determine an appropriate threshold. Figure 9 shows a sample plot of radiance values for one pixel. Once you find the frame with the maximum value for each pixel, the corresponding projector light stripe and the projector origin form a plane. Intersect the camera ray with the light stripe plane to compute the 3D point.

After this step you should have a 3D point cloud corresponding to a range image for each view. To reduce the number of spurious points, you can manually specify an axis aligned bounding box and reject points outside the bounding box. You should save each point cloud in *ply* or *obj* format such that you can load it using MeshLab. After computing point clouds for all views, you need to examine them in MeshLab and see if there are any outliers. You can manually select and delete points that are not part of the model. Click Select Vertices icon as shown in Figure 10. Then select the noisy vertices and press *Ctrl+Delete*. Figure 11 shows the point cloud before and after cleaning the outliers.

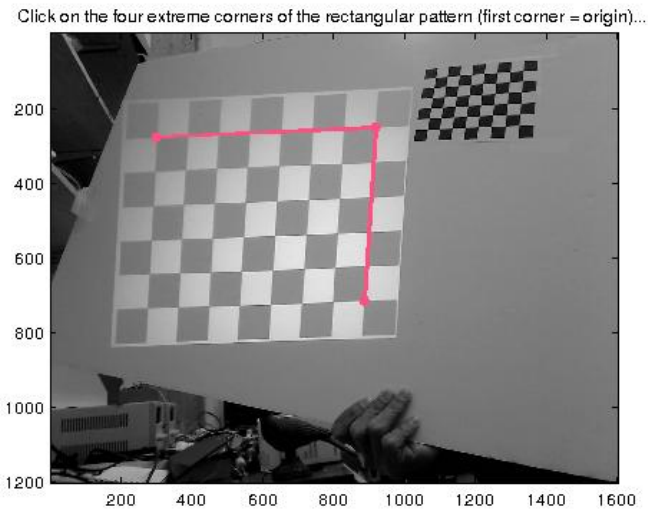

Figure 8: Clicking projected image corners.

7 Computing Range Surfaces

You will use MeshLab for all remaining geometry processing. First, you need to convert each range image to a range surface. You should take the following steps:

1. Load a point cloud

2. Compute normals
   Go to: Filters → Normals Curvature and Orientation → Compute normals for point sets

   Use at least 20 for number of neighbors. You can use more to get smoother normals. Check flip normals w.r.t. viewpoint. Leave the Viewpoint pos as $(0, 0, 0)$ because the camera is at the origin.

   Go to: Render → Show vertex normals to see if the normals are mostly correct.

   You can also click the light bulb button to see a rendering with the normals.

3. Poisson surface reconstruction
   Go to: Filters→Point Set→Surface Reconstruction:Poisson.

   Use 12 for Octree Depth and 7 for Solver Divide. Feel free to explore different parameter values. Figure 12 shows a point cloud rendered with normals and the reconstructed surface.

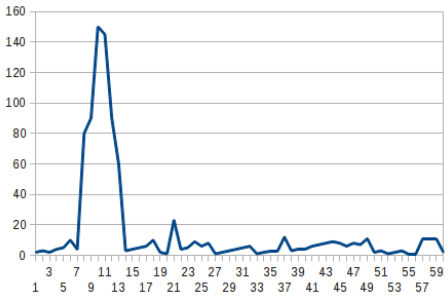

Figure 9: A clean signal for a pixel.

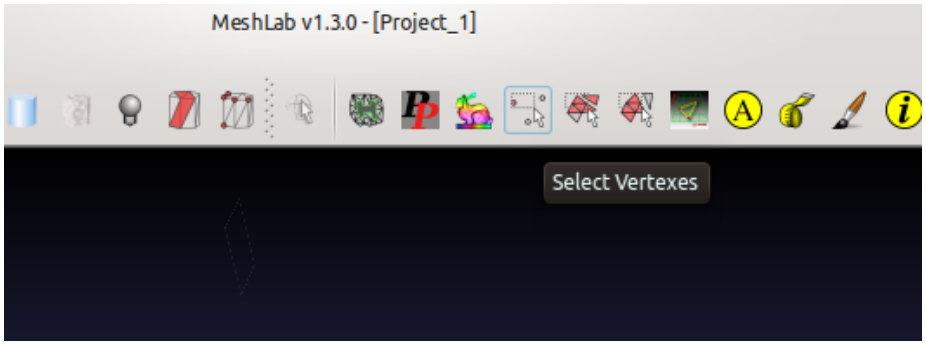

Figure 10: "Select Vertexes" icon.

4. Removing unnecessary triangles generated by the reconstruction
   Go to: Filters→ Sampling →Hausdorff Distance.

   The source mesh is a reconstructed mesh, and the target mesh is the original point cloud. Check sample surface, sample vertices and sample edges. Use 1M point samples. Again, you can explore different parameter settings. The distance is shown in the Layer Dialog. It can be displayed by clicking the icon as shown in Figure 13

5. Filters→ Selection →Select Faces by Vertex Quality.

   Set Min Quality to 0, Max Quality to about 1.5. Then, go to: Filters→ Selection →Invert Selection.

   Hit *Ctrl+Delete* to remove the unnecessary triangles. Figure 14 shows the selection process.

Apply the same procedure for each point cloud. You can automate many operations using MeshLab server mode.

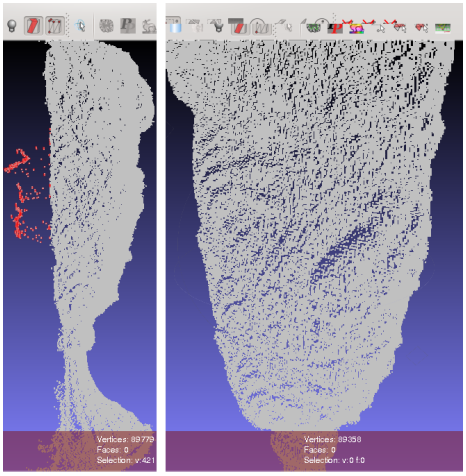

Figure 11: Before and after cleaning outliers.

8 3D Registration of Range Surfaces and Surface Reconstruction

You should take the following steps:

1. Load all your range surfaces into MeshLab.

2. Save the project (MeshLab occasionally crashes). If it crashes when you save the project, you may need to run a different version of MeshLab or run it on a different platform.

3. Click the Align icon.
   You can follow this tutorial: (`http://www.youtube.com/watch?v=4g9Hap4rX0k`) to align all the meshes. Search for "MeshLab align meshes" on your own if the video link does not work. Remember to save your work occasionally. The workflow requires some initial manual alignment by selecting a few (about 7) point correspondences. While you compute the 3D registration, you can go back to edit mode and delete low quality portions of each mesh. When you click the align icon again to continue aligning, remember to click Glue Here Mesh for meshes you already aligned.

4. Once all meshes are aligned, Right click any layer and select Flatten Visible Layers.

5. Run Poisson surface reconstruction again to obtain the final mesh. You can apply further manual cleaning if necessary.

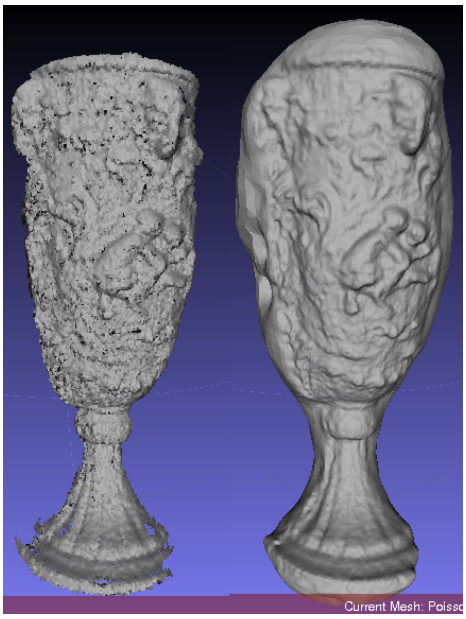

Figure 12: A point cloud with normals and its surface reconstruction.

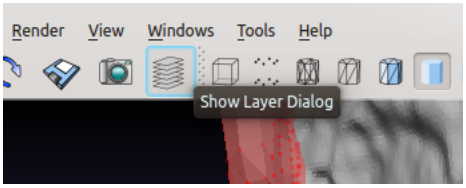

Figure 13: Layer Dialog icon.

9 Hints

1. Choose an object that you plan to scan carefully. The object should have proper size with distinctive features. It should not be transparent, too reflective or too dark.

2. Place your object properly. If it is too far away, your accuracy will decrease.. Place the object, roughly in the middle of the working volume.

3. Take slightly more scans than you think what is necessary. The registration algorithm is more robust when there is a significant overlap between two surfaces.

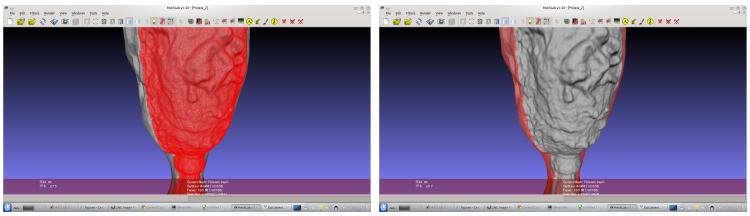

Figure 14: Selecting good triangles and inverting the selection.

10 Report

You should turn in a report describing your results. Include screenshots of the intermediate steps (e.g., calibration, range images). Include a 3D model of the object you have acquired in the PLY format.

11 Optional Features

1. Use Gray codes to accelerate scanning.
2. Recover a texture map.