ENGN 2911 I: 3D Photography and Geometry Processing
Assignment 2: Structured Light for 3D Scanning

Instructor: Gabriel Taubin[*]
Assignment written by: Douglas Lanman[†]

26 February 2009

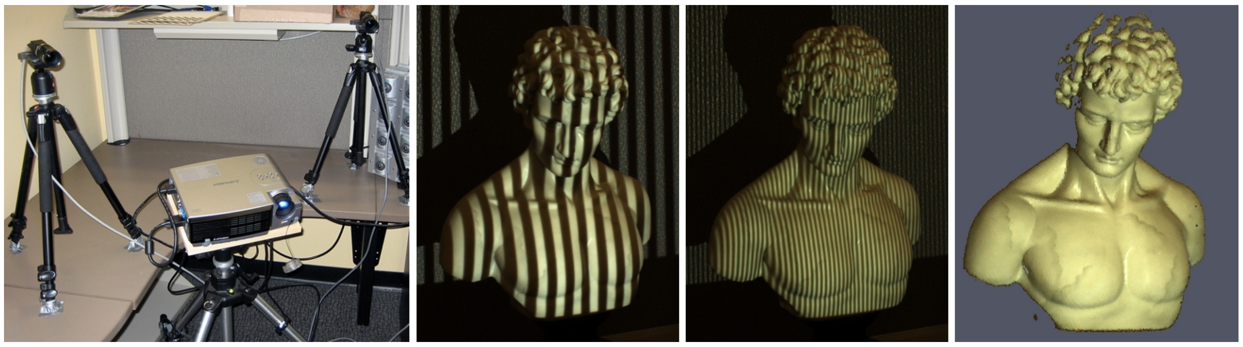

Figure 1: Structured Light for 3D Scanning. From left to right: a structured light scanning system containing a pair of digital cameras and a single projector, two images of an object illuminated by different bit planes of a Gray code structured light sequence, and a reconstructed 3D point cloud.

Introduction

The goal of this assignment is to build a 3D scanner using one or more digital cameras and a single projector. While the "desktop scanner" [2] implemented in the previous assignment is inexpensive, it has limited practical utility. The scanning process requires manual manipulation of the stick, and the time required to sweep the shadow plane across the scene limits the system to reconstructing static objects. Manual translation can be eliminated by using a digital projector to sequentially display patterns (e.g., a single stipe translated over time). Furthermore, various *structured light* illumination sequences, consisting of a series of projected images, can be used to efficiently solve for the camera pixel to projector column (or row) correspondences.

In this assignment you will be directly extending the algorithms and software developed in the previous homework. Similar to the "desktop scanner", reconstruction will be accomplished using ray-plane triangulation. The key difference is that correspondences will now be established by decoding certain structured light sequences. We have provided a variety of data sets to compare the performance and assess the limitations of each sequence.

1 Data Capture

This assignment does not require you to construct the actual scanning apparatus. Instead, we have provided you with multiple test sequences collected using our implementation. As shown in Figure 1, the system contains a single Mitsubishi XD300U DLP projector and a pair of Point

[*]taubin@mesh.brown.edu
[†]dlanman@brown.edu

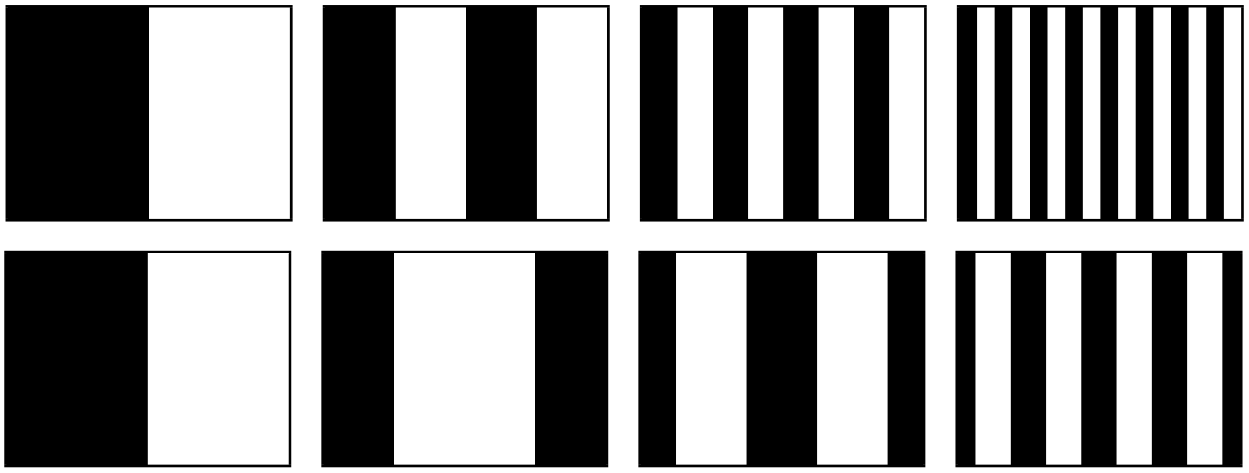

Figure 2: Structured light illumination sequences. (Top row, left to right) The first four bit planes of a binary encoding of the projector columns, ordered from most to least significant bit. (Bottom row, left to right) The first four bit planes of a Gray code sequence encoding the projector columns.

Grey GRAS-20S4M/C Grasshopper video cameras. The projector and cameras are positioned to provide sufficient reconstruction baselines for ray-plane triangulation. The projector is capable of displaying 1024×768 24-bit RGB images at 50-85 Hz [3]. The cameras capture 1600×1200 24-bit RGB images at up to 30 Hz [5]. The data capture was implemented in Matlab. The cameras were controlled using custom wrappers for the FlyCapture SDK [6], and fullscreen control of the projector was achieving using the Psychophysics Toolbox [8]. Data collection will be demonstrated in class, and we encourage students to bring their own objects to add to the test database.

The primary benefit of introducing the projector is to eliminate the mechanical motion required in "swept-plane" scanning systems (e.g., laser striping or the "desktop scanner" from the previous assignment). Assuming minimal lens distortion, the projector can be used to display a single column (or row) of white pixels translating against a black background; thus, 1024 (or 768) images would be required to assign the correspondences between camera pixels and projector columns (or rows) in our system. After establishing the correspondences and calibrating the system, a 3D point cloud could be reconstructed using ray-plane triangulation. However, a simple swept-plane sequence does not fully exploit the projector. Since we are free to project arbitrary 24-bit color images, one would expect there to exist a sequence of coded patterns, besides a simple translation of a single stripe, that allow the camera-projector correspondences to be assigned in relatively few frames. In general, the identity of each plane can be encoded spatially (i.e., within a single frame) or temporally (i.e., across multiple frames), or with a combination of both spatial and temporal encodings. There are benefits and drawbacks to each strategy. For instance, purely spatial encodings allow a single static pattern to be used for reconstruction, enabling dynamic scenes to be captured. Alternatively, purely temporal encodings are more likely to benefit from redundancy, reducing reconstruction artifacts. A comprehensive assessment of such codes was presented by Salvi et al. [10].

In this assignment we will focus on purely temporal encodings. While such patterns are not well-suited to scanning dynamic scenes, they have the benefit of being easy to decode and are robust to surface texture variation, producing accurate reconstructions for static objects (with the normal prohibition of transparent or other problematic materials). A simple binary structured light sequence was first proposed by Posdamer and Altschuler [7] in 1981. As shown in Figure 2, the binary encoding is simply a sequence of binary images in which each frame represents a single bit plane of the binary representation of the integer indices for the projector columns (or rows). For

```
BIN2GRAY(B)
1  $n \leftarrow length[B]$
2  $G[1] \leftarrow B[1]$
3  for $i \leftarrow 2$ to $n$
4      do $G[i] \leftarrow B[i-1]$ xor $B[i]$
5  return $G$
```

```
GRAY2BIN(G)
1  $n \leftarrow length[G]$
2  $B[1] \leftarrow G[1]$
3  for $i \leftarrow 2$ to $n$
4      do $B[i] \leftarrow B[i-1]$ xor $G[i]$
5  return $B$
```

Table 1: Pseudocode for converting between binary and Gray codes. (Left) BIN2GRAY accepts an n-bit Boolean array, encoding a decimal integer, and returns the Gray code G. (Right) Conversion from a Gray to a binary sequence is accomplished using GRAY2BIN.

example, column 546 has a binary representation of 1000100010 (ordered from the most to the least significant bit). Similarly, column 546 of the binary structured light sequence has an identical bit sequence, with each frame displaying the next bit.

Considering the projector-camera arrangement as a communication system, then a key question immediately arises; what binary sequence is most robust to the known properties of the channel noise process? At a basic level, we are concerned with assigning an accurate projector column/row to camera pixel correspondence, otherwise triangulation artifacts will lead to large reconstruction errors. Gray codes were first proposed as one alternative to the simple binary encoding by Inokuchi et al. [4] in 1984. The *reflected binary code* was introduced by Frank Gray in 1947 [11]. As shown in Figure 2, the Gray code can be obtained by reflecting, in a specific manner, the individual bit-planes of the binary code. Pseudocode for converting between binary and Gray codes is provided in Table 1. For example, column 546 has a Gray code representation of 1100110011, as given by BIN2GRAY. The key property of the Gray code is that two neighboring code words (e.g., neighboring columns in the projected sequence) only differ by one bit (i.e., adjacent codes have a Hamming distance of one). As a result, the Gray code structured light sequence tends to be more robust to decoding errors than a simple binary encoding.

What to turn in: Implement a function named `bincode` to generate a binary structured light sequence. The inputs to this function should be the width w and height h of the projected image. The output should be a sequence of $2\lceil\log_2 w\rceil + 2\lceil\log_2 h\rceil + 2$ uncompressed images; the first two images should consist of an all white and an all black image, respectively. The next $2\lceil\log_2 w\rceil$ images should contain the bit planes of the binary sequence encoding the projector columns, interleaved with the binary inverse of each bit plane (to assist in decoding). The last $2\lceil\log_2 h\rceil$ images should contain a similar encoding for the projector rows. Implement a similar function named `graycode` to generate the Gray code structured light sequence. Use your program to generate binary and Gray encodings for a projector with a resolution of 640×480.

2 Image Processing

The algorithms used to decode the structured light sequences described in the previous section are relatively straightforward. For each camera we must decide whether a given pixel is directly illuminated by the projector in each displayed image. If it is illuminated in any given frame, then the corresponding code bit is set high, otherwise it is set low. The decimal integer index of the corresponding projector column (or row) can then be recovered by decoding the received bit sequences for each camera pixel. In order to determine whether a given pixel is illuminated, we must assign a threshold. For instance, $2\lceil\log_2 w\rceil + 2$ images could be used to encode the projector

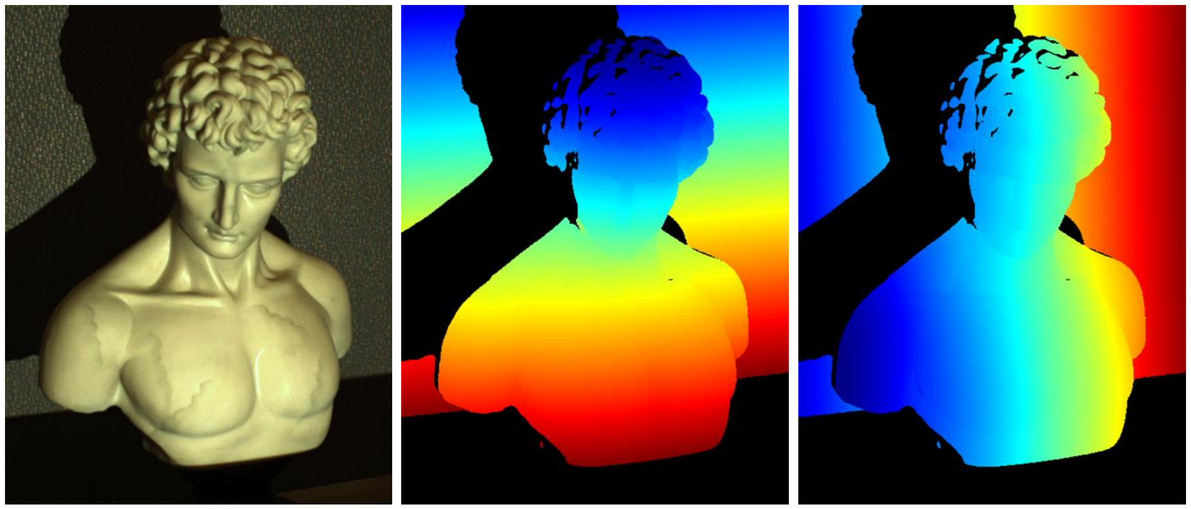

Figure 3: Decoding structured light illumination sequences. (Left) Camera image captured while projecting an all white frame. Note the shadow cast on the background plane, prohibiting reconstruction in this region. (Middle) Typical decoding results for a Gray code structured light sequence, with projector row and camera pixel correspondences represented using a `jet` colormap in MATLAB. Points that cannot be assigned a correspondence with a high confidence are shown in black. (Right) Similar decoding results for projector column correspondences.

columns, with the additional two images consisting of all white and all black frames. The average intensity of the all white and all black frames could be used to assign a per-pixel threshold; the individual bit planes of the projected sequence could then be decoded by comparing the received intensity to the threshold.

In practice, a single fixed per-pixel threshold results in decoding artifacts. For instance, certain points on the surface may only receive indirect illumination that is scattered from directly-illuminated points. In certain circumstances the scattered light may led to a bit error, in which an unilluminated point appears illuminated due to scattered light. Depending on the specific structured light sequence, such bit errors may produce significant reconstruction errors in the 3D point cloud. One solution is to project each bit plane and its inverse, as was done in Section 1. While $2\lceil \log_2 w \rceil$ frames are now required to encode the projector columns, the decoding process is less sensitive to scattered light, since a variable per-pixel threshold can be used. Specifically, we can determine whether a bit is high or low depending on whether a projected bit-plane or its inverse is brighter at a given pixel. Typical decoding results are shown in Figure 3.

As with any communication system, the design of structured light sequences must account for anticipated artifacts introduced by the communication channel. In a typical projector-camera system, decoding artifacts can be introduced from a wide variety of sources, including projector or camera defocus, scattering of light from the surface, and temporal variation in the scene (e.g., varying ambient illumination or a moving object). We have provided a variety of data sets for testing your decoding algorithms. In particular, the `man` sequence has been captured using both binary and Gray code structured light sequences. Furthermore, both codes have been applied when the projector is focused and defocused at the average depth of the sculpture. We encourage you to study the decoding artifacts produced under these circumstances in your analysis.

What to turn in: Implement a function named `bindecode` to decode the binary structured light sequences provided in the support code. The input to the function should be the directory containing the sequences encoded using your solution from Section 1. The output should be a pair of unsigned 16-bit grayscale images containing the decoded decimal integers corresponding to the projector column and row that illuminated each camera pixel (see Figure 3). Use zero to indicate that a given pixel cannot be assigned a correspondence, and label the projector columns and rows starting from one. Implement a similar function named `graydecode` to decode the Gray code structured light sequences. Turn in a plot of the decoded correspondences for the binary and Gray code sequences, for both for the focused and defocused cases, for the `man` sequences. Note that our implementation of the Gray code is shifted to the left, if the number of columns (or rows) is not a power of two, such that the projected patterns are symmetric about the center column (or row) of the image. The specific projected patterns are stored in the `/data` directory, and the sample script `slDisplay` can be used to load and visualize the provided data sets.

3 Calibration

Calibration of the cameras is accomplished using the Camera Calibration Toolbox for Matlab [1], following the approach used in the previous assignment. A sequence of 15 views of a planar checkerboard pattern, composed of 38mm$\times$38mm squares, is provided in the /calib/cam/v1 directory. Also, if you decide to apply color calibration, a sequence of images of a planar pattern for all red, blue, and green projected illumination is also provided. The intrinsic and extrinsic camera calibration parameters, in the format specified by the toolbox, are provided in the /calib/cam/v1/Calib_Result.mat.

Projector calibration, while not a fully-developed component of the Camera Calibration Toolbox for Matlab, can be implemented by extending its functionality. In our implementation, we use the popular method of projector calibration in which projectors are modeled as *inverse cameras* [9]. We assume an ideal camera is modeled as a pinhole imaging system, with real-world cameras containing additional lenses. In this model, a camera is simply a device that measures the irradiance along incident optical rays. The inverse mapping, from pixels to optical rays, requires calibrating the intrinsic and extrinsic parameters of the camera, as well as a lens distortion model. A projector can be seen as the inverse of a camera, in which a certain irradiance is projected along each optical ray, rather than measured. Once again, an ideal projector can be modeled as a pinhole imaging system, with real-world projectors containing additional lenses that introduce distortion. A similar intrinsic model, with a principal point, skew coefficients, scale factors, and focal lengths can be applied to projectors. As a result, we apply a similar calibration pipeline as used for cameras.

To calibrate a projector, we assume that the user has a calibrated camera. All that is required to calibrate the projector is a diffuse planar pattern with a small number of printed fiducials located on its surface. In our design, we used a piece of poster board with four printed checkerboard corners. As shown in Figure 4, a single image of the printed fiducials is used to recover the implicit equation of the calibration plane in the camera coordinate system (e.g., using `extrinsicDemo` from the previous assignment). A checkerboard pattern is then displayed using the projector. The 3D coordinate for each projected checkerboard corner is then reconstructed. A set of 2D to 3D correspondences are then used to estimate the intrinsic and extrinsic calibration from multiple views of the planar calibration object.

In this assignment you are not required to implement projector calibration. We have included 20 images of the projector calibration object in the /calib/proj/v1 directory. The printed fiducials were horizontally separated by 406mm and vertically separated by 335mm. The camera and projector calibration can be loaded from /calib/calib_results/calib_cam_proj.mat. Note that the intrinsic and extrinsic parameters are in the format used in the Camera Calibration Toolbox for Matlab. The `slCalib` function can be used to visualize the calibration results.

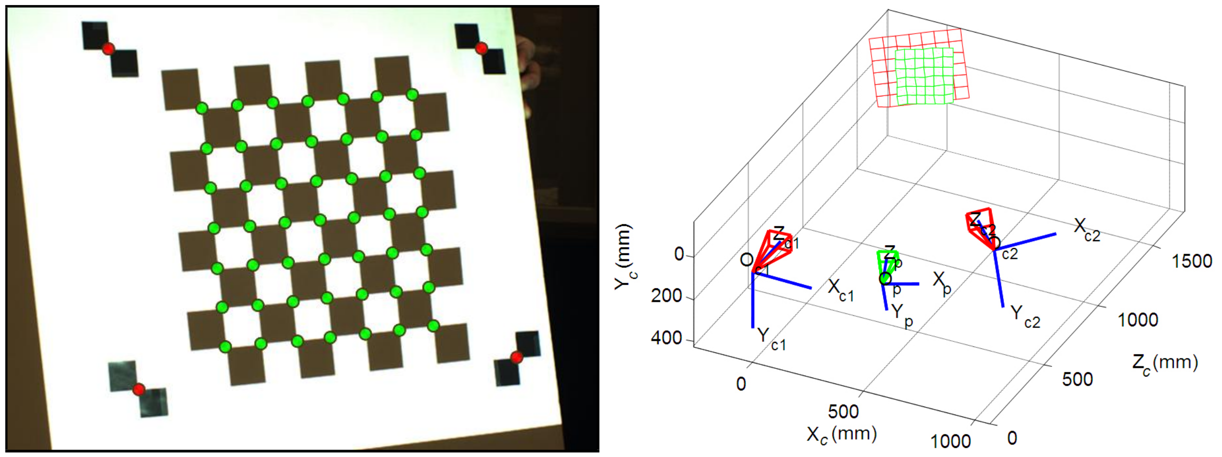

Figure 4: Projector calibration. (Left) A planar calibration object with four printed checkerboard corners is imaged by a camera. A projected checkerboard is displayed in the center of the calibration plane. The physical and projected corners are manually detected and indicated with red and green circles, respectively. (Right) The extrinsic camera and projector calibration is visualized using `slCalib`. Viewing frusta for the cameras are shown in red and the viewing frustum for the projector is shown in green. Note that the reconstruction of the first image of a single printed checkerboard, used during camera calibration, is shown with a red grid, whereas the recovered projected checkerboard is shown in green. Also note that the recovered camera and projector frusta correspond to the physical configuration shown in Figure 1.

What to turn in: Implement a function `campixel2ray` to convert from camera pixel coordinates to an optical ray expressed in the coordinate system of the first camera. Implement a similar function `projpixel2ray` to convert from projector pixel coordinates to an optical ray expressed in the common coordinate system of the first camera. Finally, implement a pair of functions called `projcol2plane` and `projrow2plane` to convert from projected column and row indices, respectively, to an implicit parametrization of the plane projected by each projector column and row in the common coordinate system. Include a plot, possibly extended from `slCalib`, illustrating your solution. For extra credit, we encourage you to implement your own projector calibration routines.

4 Reconstruction

The decoded set of camera and projector correspondences can be used to reconstruct a 3D point cloud. Several reconstruction schemes can be implemented using the sequences described in Section 1. The projector column correspondences can be used to reconstruct a point cloud using ray-plane triangulation. A second point cloud can be reconstructed using the projector row correspondences. Finally, the projector pixel to camera pixel correspondences can be used to reconstruct the point cloud using ray-ray triangulation (i.e., by finding the closest point to the optical rays defined by the projector and camera pixels). A simple per-point RGB color can be assigned by sampling the color of the "all on" camera image for each 3D point. Reconstruction artifacts can be further reduced by comparing the reconstruction produced by each of these schemes.

What to turn in: Implement a function named `slReconstruct` to reconstruct 3D point clouds for the sequences provided in the support code. Describe your reconstruction algorithm, especially any

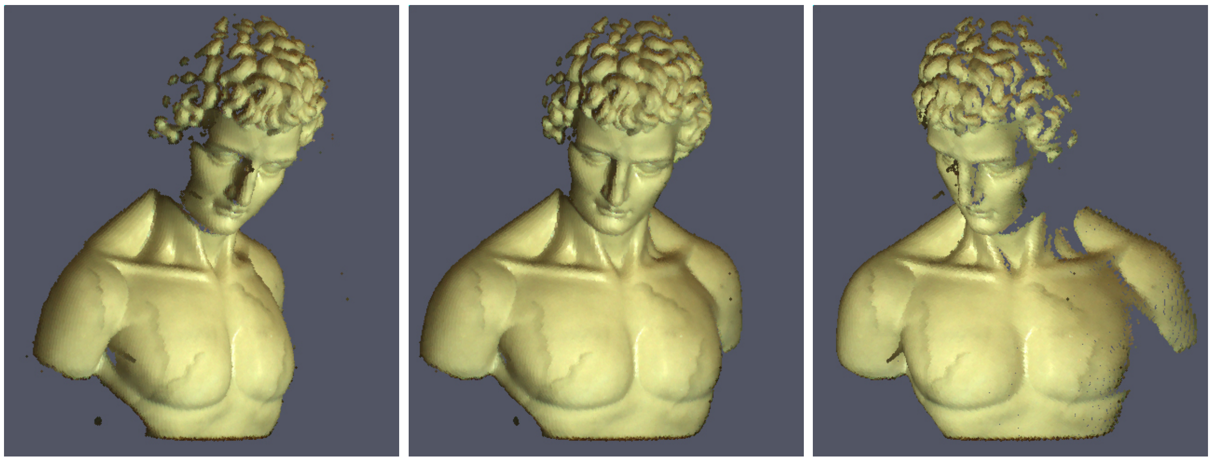

Figure 5: Reconstruction results for the man sequence using a focused Gray code.

procedures used to eliminate outliers, improve color accuracy, or otherwise reduce artifacts. Turn in a VRML file for each data set, containing a single indexed face set with an empty coordIndex array and a per-vertex color (similar to the VRML files you created in the previous assignment).

5 Post-processing and Visualization

The structured light scanner produces a 3D point cloud. Only points that are both imaged by a camera and illuminated by a projector can be reconstructed. As a result, a complete 3D model of an object would typically require merging multiple scans obtained by moving the scanning apparatus or object (e.g., by using a turntable). These are topics for a later assignment, however we encourage you to implement your solution so that measurements from multiple cameras, projectors, and 3D point clouds can be merged in the future. For this assignment we will use point-based rendering to display the acquired data using any VRML viewer that supports point rendering (see Figure 5).

What to turn in: Use the provided VRML viewer to save images of your reconstructed point clouds. Turn in at least one reconstructed image, shown from a different viewpoint than any camera image, for each provided sequence. Use these images to assess the benefits and limitations of the binary and Gray code structured light sequences. Explain what modifications could be made to improve the overall performance of the structured light scanner implemented in this assignment.

Submission Instructions

You should submit clear evidence that you have successfully implemented the "structured light scanner". In particular, you should submit: (1) an archive named `3DPGP-HW2-Lastname.zip` containing your source code without the `/calib`, `/codes`, or `/data` directories, (2) a typeset document explaining your implementation, and (3) a set of reconstructed point clouds as VRML files. Make sure to answer the questions posed at the end of each section. If you didn't use Matlab, please provide brief compilation instructions. Include a `README` file with a short description of every file (except those we provided) in `3DPGP-HW2-Lastname.zip`. Final solutions should be emailed to the instructor at `taubin@mesh.brown.edu`. Please note that we reserve the right to request a 15 minute demonstration if the submitted documentation is insufficient to assess your solution.

References

[1] Jean-Yves Bouguet. Camera calibration toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[2] Jean-Yves Bouguet and Pietro Perona. 3d photography on your desk. `http://www.vision.caltech.edu/bouguetj/ICCV98/`.

[3] Mitsubishi Electric Corp. XD300U user manual. `http://www.projectorcentral.com/pdf/projector_manual_1921.pdf`.

[4] S. Inokuchi, K. Sato, and F. Matsuda. Range imaging system for 3-d object recognition. In *Proceedings of the International Conference on Pattern Recognition*, 1984.

[5] Point Grey Research, Inc. Grasshopper IEEE-1394b digital camera. `http://www.ptgrey.com/products/grasshopper/index.asp`.

[6] Point Grey Research, Inc. Using Matlab with point grey cameras. `http://www.ptgrey.com/support/kb/index.asp?a=4&q=218`.

[7] J.L. Posdamer and M.D. Altschuler. Surface measurement by space encoded projected beam systems. *Computer Graphics and Image Processing*, 18:1–17, 1982.

[8] Psychophysics Toolbox. `http://psychtoolbox.org`.

[9] R. Raskar, M. S. Brown, R. Yang, W.-C. Chen, G. Welch, H. Towles, B. Seales, and H. Fuchs. Multi-projector displays using camera-based registration. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 161–168. IEEE Computer Society Press, 1999.

[10] J. Salvi, J. Pages, and J. Batlle. Pattern codification strategies in structured light systems. In *Pattern Recognition*, volume 37, pages 827–849, April 2004.

[11] Wikipedia. Gray code. `http://en.wikipedia.org/wiki/Gray_code`.