Computational Photography and Capture Class Project : Structured Light Depth Acquisition

Hugo Germain

March 10^{th} 2017

Introduction

In this Coursework, we will be implementing a Structured Light Depth Acquisition algorithm. First, we will be studying 3D point reconstruction from a set of images using Structured Lights, given a Camera and Projector calibration. Then, we will be using calibration images to compute the Camera and Projector intrinsic and extrinsic matrices. Lastly, we will be using our own captured data as well as some real provided data, to perform the steps described above, and discuss our results.

1 3D Reconstruction

First, we will be implementing a 3D Reconstruction algorithm, based on the paper *High Accuracy Stereo Depth Maps Using Structured Light*, by Daniel Scharstein and Richard Szeliski. In this part, we will be first using the provided synthetic scenes. The full algorithm can be found in the `RECONSTRUCTION.m` Matlab file.

Performing 3D reconstruction from stereo cameras is a well-studied problem. One of the main issues with using a classic correspondence-point approach is that we often lack features on planar or uniform surfaces, and thus fail to get depth information. Instead, one can perform 3D reconstruction using "active" stereo, namely Structured Lighting. The idea is to project a pattern that will define points *uniquely* and allow for a much denser and robust point cloud.

1.1 Retrieving (u,v) codes

The pattern we use is a set of Gray Codes. Using a stack of 20 images, we can compute a unique binary code for each point in the image. We also use 20 additional "negative" images to allow for unreliable point detection (see 1.2).

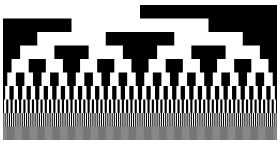


FIGURE 1 – Gray Code base used for Structured Lighting

In Matlab, we decode the (u,v) code for each pixel by retrieving the decoded Gray Code using the stack of 40 images. For each Gray Code level, we set the corresponding binary value to 1 if :

$$G^+_{i,j} - G^-_{i,j} > 0 \text{ (G^+ and G^- being the positive and negative codes)}$$

Refer to `compute_code.m` for more details on the Matlab implementation.

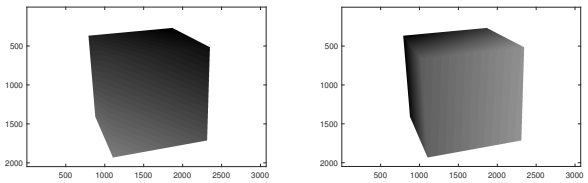


FIGURE 2 – (u,v) codes obtained for the "cube_T1" sequence

Because the level of details in the lowest Gray Code pattern is smaller than the resolution of the camera, we observe that neighbouring pixels in the photographed sequences have equal values. We can see that as a first limitation of the Structured Light method. Indeed, getting equal (u,v) codes for neighbouring pixels will lead to an equal depth, hence small planar patches in the 3D reconstructed point cloud.

One alternative would be to smooth the (u,v) codes vertically and horizontally respectively, using for instance a gaussian convolution. However, we can already imagine that on real data and due to effects like aliasing and a lower resolution projector, this effect will be harder to compensate for.

1.2 Eliminating unreliable (u,v) codes

To get rid of pixels whose gray code cannot be determined reliably, we follow the method described in *High Accuracy Stereo Depth Maps Using Structured Light*. First, we compute the sum of distances $G^+_{i,j} - G^-_{i,j}$ for each Gray Code level. If this sum is found to be below a certain Threshold T, then it means that the surface on which we measure the luminosity difference is either too specular or not diffusing the light properly, and we discard the studied pixel.

1.3 Computing the Depth

Now that we have the correspondence between the projector and camera points $\left(\begin{pmatrix} u \\ v \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix}\right)$, we can triangulate the 3D points using the Extrinsic and Intrinsic matrices by minimising the squared distance :

$$\hat{\boldsymbol{w}} = \underset{w}{\operatorname{argmin}} \left(\sum_{j=1}^{J} (\boldsymbol{x}_j - \text{pinhole}[\boldsymbol{w}, \boldsymbol{\Lambda_j}, \boldsymbol{\Omega_j}, \boldsymbol{\tau_j}])^T (\boldsymbol{x}_j - \text{pinhole}[\boldsymbol{w}, \boldsymbol{\Lambda_j}, \boldsymbol{\Omega_j}, \boldsymbol{\tau_j}]) \right)$$

In practice, we solve this problem by solving the linear system described in Simon Prince's book, as referred in Lab 2 of this module. Instead of having two cameras, we assume that the second camera is instead the projector and use its extrinsic and intrinsic matrices. Once we have solved the linear system using a least-squared method, we make sure to reproject our world points in the camera space, by multiplying $\boldsymbol{w}$ by the camera Extrinsic matrix.

Here is the depth map we get (computed as the third coordinate of reprojected $\boldsymbol{w}$) :

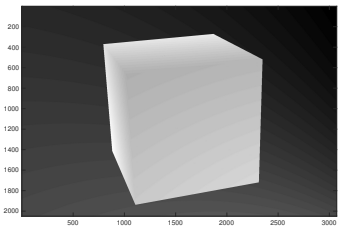


FIGURE 3 – Depth Map, computed using `compute_depth.m`

1.4 Visualising the 3D Point Cloud

For a better understanding of the obtained point cloud, we export the 3D points to a .PLY file, to then open it in MeshLab. Refer to `save_ply.m` for the PLY file exportation. The files are then saved as *<file_name>.txt*. To open it in MeshLab, one must choose to skip the first 7 rows, and choose *space* as the separator.

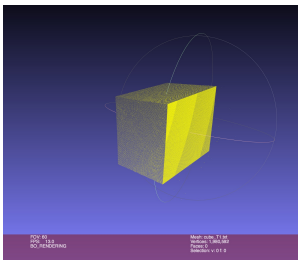
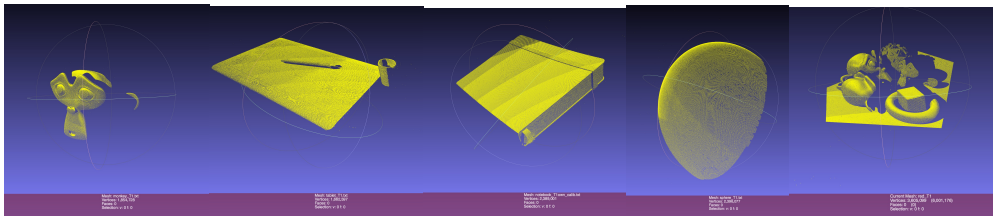


FIGURE 4 – 3D Point Cloud, visualized in Mesh Lab ('cube_T1.txt')

1.5 Results

Overall we get some rather satisfying results. The point clouds are reconstructed with a good level of details and with very little to no noise. However, we can see that this method leads to a rather strong occlusion. Having multiple projections from different angles could allow for more coverage of the scene.



(a) *monkey_ T1* (b) *tablet_ T1* (c) *notebook_ T1* (d) *sphere_ T1* (e) *red_ T1*

FIGURE 5 – More results from other synthetic scenes

Besides, the projector and camera calibration seem to be leading to some small perspective distortions. On scenes like the one with the Monkey, which is more colorful, we could expect more difficulties to extract 3D information from dark surfaces, but it is not the case. Even on very complex and specular surfaces like the one named 'red', the 3D reconstruction perform very well, and for the notebook we can even distinguish the presence of hair.

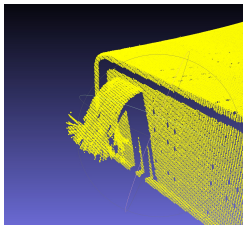


FIGURE 6 – Hair details on ('notebook_T1.txt')

2 Camera calibration

In this section, we will be implementing the computation of the Intrinsic and Extrinsic Matrices $\mathbf{\Lambda}$ and $\mathbf{\Omega}$ for both the Camera and the Projector. Calibrating the camera is done in a similar fashion as in Lab 2 of this course, using a set of checkerboard images. Calibrating the Projector requires the computation of the same set of images, "seen" by the projector.

The Camera Calibration can be performed in Matlab by first computing the reprojected images by running `CALIBRATION.m`. Then, using the `calib_gui` program and the obtained images, we estimate the intrinsic and extrinsic matrices.

2.1 Reprojecting the images

Based on the provided checkerboard patterns, we determine the projection matrices for the projector and the camera using the technique described in *Projector calibration for 3D scanning using virtual target images*, by Hafeez Anwar, Irfanud Din and Kang Park. Using the projected patterns seen by the camera, we compute the homography that will send the projected checkerboard to the reference coordinates (provided in the Coursework). The homography estimation is done by solving a least-square linear problem. We base our code on the Machine Vision Practical on homographies (see `CALIBRATION.m`) for more details. Once our homography is estimated, we simply apply the transformation to the non-projected checkerboard (seen by the camera), and we obtain the checkerboard in the projector space. The obtained images can be found under '*calibration_jpg*'

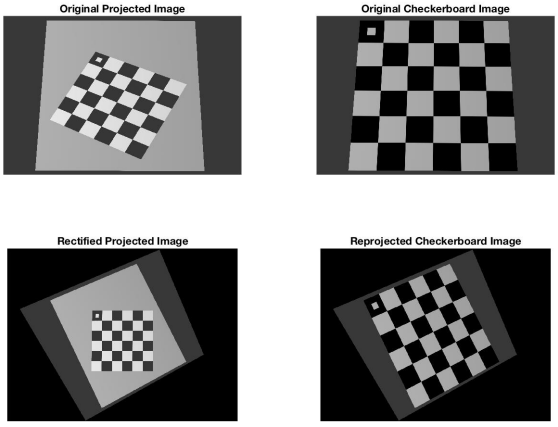


FIGURE 7 – On the first column, we have the reference (projected) checkerboard, and its transformation once we apply the estimated homography. As expected, the checkerboard lands on the reference points. On the second column, we have the photographed checkerboard and its rectified version using the previously estimated transformation.

In order to check for our reprojection more precisely, we can compute the projection matrix as the product of the Intrinsic and Extrinsic matrix. If we compute this matrix for the projector and apply it to photographed checkerboard, we should be able to retrieve the same reprojected checkerboard (in the projector space). We can then proceed similarly with the camera projection matrix and apply it to the previously reprojected image, as we should retrieve our original photographed image.

Unfortunately, I was not able to figure out how to apply a 4x4 transformation matrix to an image in Matlab without errors occuring. Hence, I was not able to check for consistency between the provided calibration matrices and those estimated further below.

2.2 Estimating the Extrinsic and Intrinsic Matrices

Now that we have our pair of images, we can use them to estimate the Extrinsic and Intrinsic Matrices. To do so, we make use of the `calib_gui` program, as we did in Lab 2. To compute the intrinsic matrix, we treat the set of images for the camera and the projector independently.

```
KK =
   1.0e+03 *
    4.8006         0    1.5356
         0    4.8005    1.0222
         0         0    0.0010
```

(a) Camera Intrinsics

```
KK =
   1.0e+03 *
    3.1098         0    0.7575
         0    3.2258    0.6632
         0         0    0.0010
```

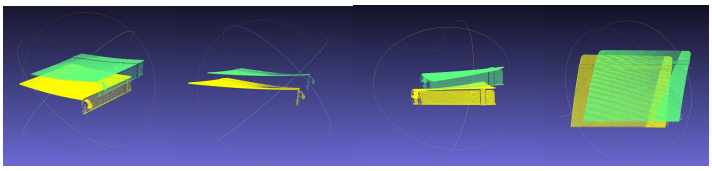
(b) Projector Intrinsics

FIGURE 8 – More results from other synthetic scenes

We get values that are roughly coherent with those provided in the calibration files. Then, we estimate the Extrinsics using a single pair of images. Because we cannot input a world coordinate frame, our Extrinsic matrix is scaled differently from the one provided in the Coursework. Hence, when computing our point cloud, the 3D reconstruction gives coherent results but scaled much differently from those visualized earlier, making the superposition of both models difficult. This may also be coming from some incoherent values inputed in when extracting the grid corners with `calib_gui`, such as the exact printed square size (especially for real data).

The 3D point clouds generated from this calibration can be found under *<filename_own_calib.txt>*.

The estimated matrices can be found under `load_synthetic_calib_results.m`. To display both meshes next to each other, we open them in MeshLab as rescale them to a Unit Box size.



(a) Cam View (b) Left View (c) Front View (d) Top View

FIGURE 9 – Notebook Meshes coming from the provided calibration (yellow) and our own estimated calibration (green)

Even though our estimated are scaled differently, we can still observe some coherent 3D reconstruction, with very little distortion, except the one observable on the top view.

The main effect of calibration is noticeable on the perspective and global geometry of the 3D point cloud. If the camera and projector are not well calibrated, we will observe distortions. A poor calibration may be due to small offsets when extracting the grid corners in `calib_gui`, as well as small distortions in the paper-printed checkerboard pattern.

More importantly, there could very possibly be calibration errors for the projector stemming from the reprojection of the calibration images. Indeed, by visualising those images we can observe slight distortions on the projected pattern, which ideally should be matching the given checkerboard coordinates perfectly.

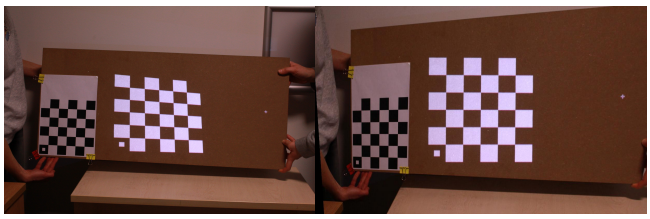
3 Real Image Sets and Own Data Capture

We now look into using real data instead of synthetic 3D images. We will both be using real data provided in the Coursework, and our own captured real-world data.

The data was captured with the help of Maud BUFFIER and Céline DUPUIS.

3.1 Capturing Real-world data

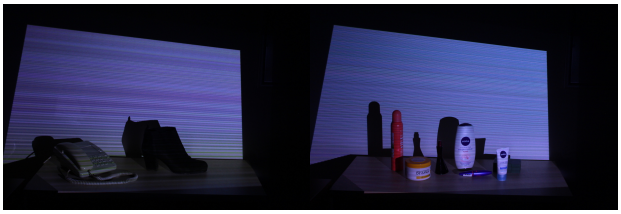
To capture the data, we used a simple DSLR camera as well as a projector. We first made sure to set the DSLR to a manual mode and manual focus, in order to avoid changes of exposure or changes in the camera intrinsics. First, we shoot some calibration images, and then project our gray-code pattern on our scene.



(a) Reference Images (b) Reprojected Checkerboard

FIGURE 10 – Real-world Captured Images

The Calibration is done in a similar fashion as previously. We project a checkboard on a plate on which we stuck a printed checkerboard. The projected checkerboard will be used to estimate the homography between the camera and the projector. To do so, we compute the precise location of the top left-hand corner of the pattern as well as its size in pixel. The printed checkerboard will be used to be reprojected using the estimated transformation, and then compute the Extrinsics and Intrinsics as previously.



(a) Shoe Scene (b) Bathroom Scene

FIGURE 11 – Gray Code patterns being projected on both scenes

We capture two sets of real-world data. The first one can be found under the folder 'shoe_T1/', and presents a scene with a landline phone and a shoe. The second one is under 'bathroom_T1/' and shows a more complex scene with more reflective objects.

3.2 Performing 3D Reconstruction

First, we perform the Camera and Projector calibration (which is done similarly as in section 2). The results can be found under `load_real_calib.m`. Then, we compute the (u,v) codes. Again, we use our gray-code patterns and compute their corresponding (u,v) value for each pixel on the image, and discard those who cannot be computed reliably.

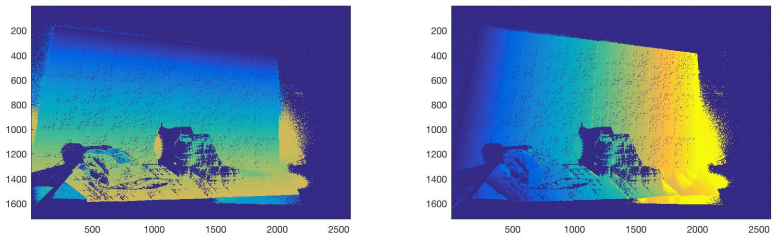


FIGURE 12 – UV codes for the Shoe Scene

A first noticeable difference with the synthetic data is that we observe phenomenon like *aliasing*. Because our projector has a limited resolution, displaying higher resolution patterns with high-frequency details will map several pixels to a single one. Thus, with very narrow stripes, the image quickly gets blurry and makes us loose this small level of details. However, the error caused is small as it only concerns high-frequency patterns, whose range map to (u,v) stripes that are only a couple of pixels wide.

Another artifact is the presence of parasite colors, due to the projection frequency. Hopefully, the latter artifact will cause less problems than the first one, as we managed to get satisfying results on the colorful synthetic data but converting it to grayscale. Lastly, some artifacts may also be coming from small vibrations on the tripod which hold either the camera or the projector, and cause an offset in the gray code pattern projections.

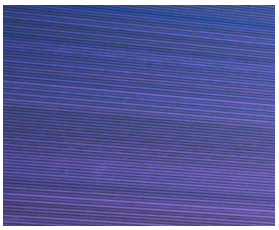
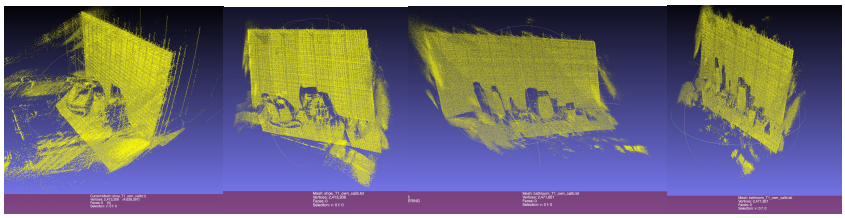


FIGURE 13 – Example of Aliasing and Colour artifacts

Then, we compute the 3D point cloud as previously, by minimizing the squared error on the 3D points. By looking at the UV codes, we can already tell that the shoe will be reconstructed poorly. Indeed, being in a dark leather it absorbs the light and prevents us from properly reading the UV codes. Here are the obtained results for both scenes :



(a) Shoe/Phone (b) Shoe/Phone (c) Bathroom (d) Bathroom

FIGURE 14 – Real-world Reconstructed Point Clouds (captured data)

With our own data, the first scene is reconstructed rather well, although we witness a perceptible perspective distortion, most likely due to a poor estimation of the Extrinsic and Intrinsic Matrices. We manage to distinguish the phone quite precisely, but the shoe - which is very dark and absorbs light - produces a noisier point cloud. We also get a fair amount of noise. This noise could be deleted by increasing the Threshold for the unreliable pixels, however this leads to loosing most of the shoe reconstruction.

The second scene presents similar artifacts. The objects in front of it being very reflective and specular, they contain quite a lot of holes and build up noise.

Using the provided real data, we obtain some highly stretched results. This most likely comes from errors in the calibration. Indeed, the UV codes look reasonable (see below), and we manage to distinguish the structure of the scene in the 3D point cloud despite it being distorted.

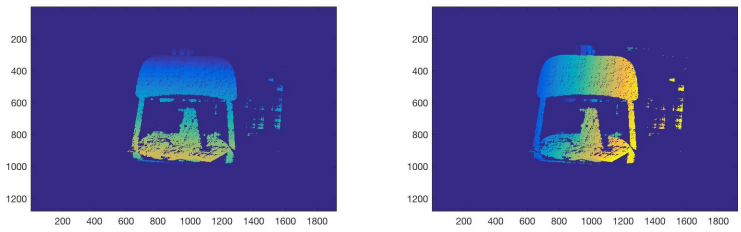
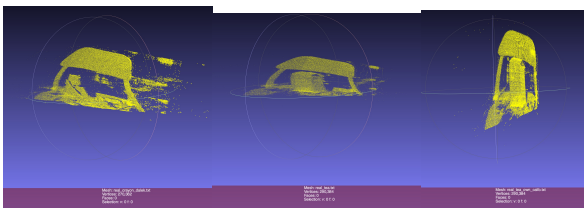


FIGURE 15 – UV codes for the Crayon Scene

Here are the obtained reconstructed 3D results :



(a) Crayon (b) Tea (c) Rescaled Tea

FIGURE 16 – Real-world Reconstructed Point Clouds (provided data)

We observe some highly stretched results. This most likely comes from the camera and projector calibration. We must state that the reprojected images used for calibration resulted in the printed checkerboard to go out of the frame, reducing the number of exploitable squares to 9.

The last figure on the right presents a rescaled version of the Tea Scene (re-scaled using MeshLab), which shows that we still have some consistency and structure in the 3D reconstruction.

Conclusion

In this Coursework, we implemented a Structured Light 3D reconstruction algorithm, from the camera calibration to the computation of the whole point clouds. We were able to see the limitations of the this method, namely the importance of a very precise calibration, as well as having high quality images and good shooting conditions to prevents artifacts. We were also able to see how some materials are hardly compatible with such an active stereo acquisition method, simply because they either absorb or reflect light too much.

References

- **High Accuracy Stereo Depth Maps Using Structured Light** by Daniel Scharstein and Richard Szeliski
- **Projector calibration for 3D scanning using virtual target images** by Hafeez Anwar, Irfanud Din and Kang Park

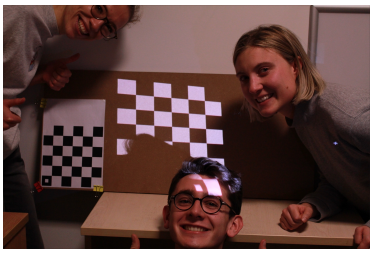


FIGURE 17 – A group of happy students