CSE586/EE554 Spring 2019 Project 1 Due Date: Tuesday Apr 9, 11:59PM

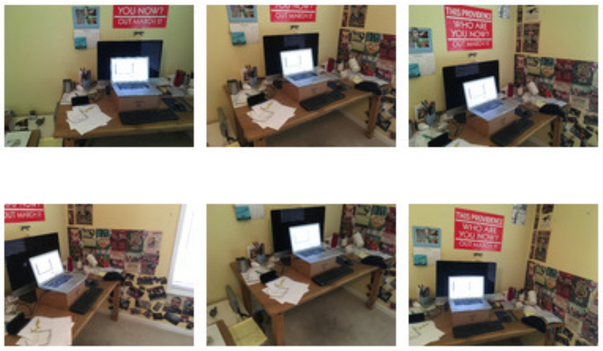
In this project you will take steps towards implementing an augmented reality viewer that displays artificial objects overlaid on images of a real 3D scene. Your approach will be offline, but it has many of the same components that, if you spent a lot of effort rewriting to run at frame rate, could form a real-time AR app like the kind enabled by the ARKit framework on iOS or ARCore on Android.

The key to this effort will be using the publicly available COLMAP software for 3D reconstruction to recover a 3D point cloud representing the structure of a scene viewed from overlapping camera views. You will use this on your own set of images. You will then take the results written out by COLMAP and do postprocessing to find the dominant plane of the scene (typically a flat horizontal surface, or a vertical wall or building façade), place a virtual 3D object on that plane, and then project it back into the original images to show what the scene would look like if that object had been there.

Except for using COLMAP to do the brunt of the work, you should implement the rest of the code yourself, without making use of computer vision library routines. Specifically, you will write your own RANSAC-based plane fitting routine, and I also want you to implement a function for 3D to 2D camera projection, even though Matlab and OpenCV and other popular packages have those in them. It is likely you will also be doing some simple geometric manipulations (e.g. rotation and translation) to map points between coordinate systems. It is OK to use numerical libraries to solve sets of equations or other general purpose math problems. You also will also need some ability to display graphics, for example plotting 3D points and drawing filled polygons in 3D, as well as drawing filled polygons on top of 2D images, and it is OK if these call out to some kind of graphics libraries. Although you can use whatever programming language you want, math and graphics is easy to do using built-in functions in Matlab (just saying).

Specific steps:

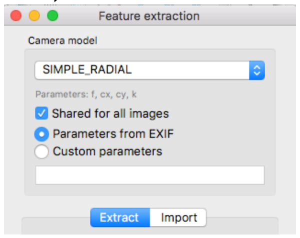
1) Collect a set of images that show the same area of a 3D scene. Your life will be easier if you take these yourself, with a single camera (cell phone camera is OK) at a single focal length (don't zoom in or out between shots) because then the internal camera parameters will be the same. Make sure your "scene" has a dominant planar surface in it, since that is where you are going to put your virtual object. This surface should have a lot of texture to make it easy for COLMAP to find and match features. Ideas for "good" surfaces with lots of corner-like textures include bulletin boards, building facades, wood tabletops, granite countertops, etc. To make your scene more interesting, try to also include some 3D structure besides the dominant plane, that is, don't just take a close up of a single planar surface. I've included an example below of images I took of a little work area in the corner of a room. The posters on the right side wall contain lots of features for matching, and that wall ends up becoming the dominant plane in my reconstructed scene.



2) Get a working version of COLMAP from https://colmap.github.io/ . There are compiled binary versions for Windows, Mac OS X, and Linux at (https://colmap.github.io/install.html). There are also a lot of other resources available on the main site, including documentation and tutorial videos. Play around with COLMAP, and look at the tutorials and docs to figure out how to do the following:

a) input your set of images from part 1,

b) run feature extraction, matching, incremental reconstruction and bundle adjustment to get a "sparse" reconstruction, which will be a 3D cloud of points. There are a lot of user settable parameters, but it is likely that the default settings will work OK (I would try that first), with the exception of: if you followed my advice and took all the pictures with the same camera, click the box "Shared for all images" for the camera model specification under feature extraction (see figure below). This will tell COLMAP to fit a single set of internal camera parameters using data from all the images, instead of using a different camera model for every image. On the other hand, if you just downloaded a set of images taken by different cameras from the internet, you should not check the box, therefore allowing COLMAP to know each image is from a different camera.


Feature extraction

Camera model

SIMPLE_RADIAL

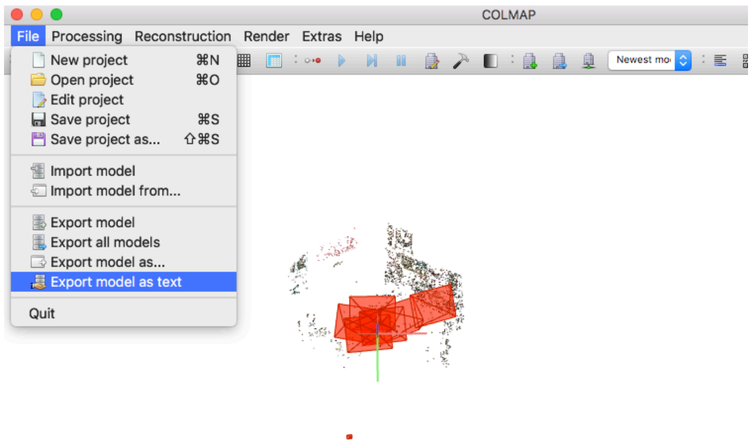
Parameters: f, cx, cy, k

[✓ Shared for all images]

[✓ Parameters from EXIF]

Custom parameters

c) export the sparse 3D point model COLMAP produces as a set of text files (see menu item selected in the figure below – by the way, these screenshots are from the Mac binary, and menus for windows or linux might differ slightly). This exporting will output a set of 3 text files: cameras.txt, images.txt, and points3D.txt . These are what you are going to use for the rest of the project. The format of the text files is documented at https://colmap.github.io/format.html



3) Read in the 3D point cloud stored in points3D.txt . We only need the X,Y,Z coordinates for each point for what we want to do, and can ignore other fields in the text file.

4) Here is the first legitimate piece of code you need to write: make a RANSAC routine to find the largest subset of 3D points that can be described by the equation of a 3D plane. To do this you need to ask yourself: What is the minimum number of 3D points needed to fit a 3D plane? How to randomly sample a minimal set of points? How to fit the equation of a plane to the points in a sample? How to determine which other points in the model lie close enough to that plane to be

considered as "inliers" that should be counted when voting for the current plane fit? How to keep track of the inliers that have voted for the largest coplanar set found so far? After a large number of RANSAC interations (e.g. several thousand), you will hopefully end up with a large set of inliers that represents the "dominant" plane in the scene, where dominant in this case means the plane that contains the most scene points.

5) Display the 3D point cloud produced by COLMAP and denote the inlier points in some way (for example you could draw them in a different color). Make sure your inlier set / dominant plane looks like what you would expect to find. Also include a picture in your project report.

Aside: If you are using Matlab, plot3 is a good function to use to plot a set of 3D points. After plotting, "rotate3D on" is also cool, as it allows you to interactively rotate the view so you can look at it from different angles.

6) This next part isn't rocket science but it could cause some difficulties if you aren't familiar with coordinate systems and geometric transformations. I'd like you to form a local x-y-z 3D coordinate system where the dominant plane you found above becomes the z=0 plane. Choose the local origin of this coordinate system so that x=0, y=0 lies roughly in the "middle" of the set of inlier points that lie on the dominant plane. Figure out the a 3D Euclidean transformation (rotation and translation) that maps an x,y,z point in your local coordinate system into scene X,Y,Z coordinates.

7) Create a virtual object to put in the scene. For simplicity, you can just make it a 3D box. Define a 3D box with bottom surface lying in your local z=0 plane, so that the center of that rectangular bottom surface is centered at x=0, y=0,z=0. Form the 8 corners of the box in x,y,z coordinates, and convert them into scene X,Y,Z coordinates using the transformation you defined in part 6. At this point, it might be helpful to visualize the box in 3D, to make sure all is well. In Matlab, you could do this by drawing the rectangular surfaces of the box in the same window where you have drawn your 3D points. This web page might be helpful: http://www.matrixlab-examples.com/3d-polygon.html . Interactively rotate the view to see if the box is where you hoped it would be (and debug if not).

At this point we have placed a virtual object within a 3D scene coordinate system that was created from a set of images. We are now going to project that virtual object into each of those images, and overlay it on the original pixel values. This is the "augmented reality" part!

8) Read in the external and internal camera parameters from files cameras.txt and images.txt . The internal parameters are in cameras.txt , and if you followed my suggestion of having COLMAP estimate just a single set of internal parameters across all images, then there is only one set of internal parameters in the file. Refer to https://colmap.github.io/format.html to see what parameters are specified and in what order. The external parameters (rotation and translation) describing camera pose for each image are in images.txt . There are a lot of other numbers in that file that we don't care about, so refer to the documentation to see which lines of the file you want to read. Note that rotation is represented as a unit quaternion. It is OK if you use a predefined routine to convert from quaternion to 3x3 rotation matrix. In Matlab, that function is quat2rotm.

9) This is the last significant piece of vision-related code you will write. Refer to our lectures on camera models and associated reference material to write your own routine that takes external and internal camera parameters and projects a given set of 3D (X,Y,Z) points into a set of 2D (col,row) pixel locations. DO NOT USE OpenCV or other vision code to do this, I want you to write it yourself. However, feel free to look at other source code if you need some guidance on how other people implement camera projection. In fact, it might be a good idea to find the camera projection function

in COLMAP that matches your kind of camera model (e.g. SIMPLE_RADIAL) since that will exactly define what the 3D to 2D projection model is for the set of parameters you have.

10) Finally, for each of your original images, project the corners of your 3D virtual box object into the image and draw it overlaid over the original pixels values. You should at least draw a wireframe representing the edges of the box. However, the visualization will be more compelling if you draw filled polygons for each projected surface of the box, so that pixels representing parts of the scene that are behind the box from a particular camera viewpoint will be hidden from view. It's actually a little nontrivial to do this, since you also want faces of the box that are closer to you to hide faces of the box that are farther away. I believe that for a nice convex shape like a box, you can achieve this effect by drawing the faces of the box in order from farthest away to closest, determined by figuring out distance along the principle axis OF THE CAMERA (aka depth) to each of the corner points of the box, and ordering the faces by min depth to any of the 4 points bounding each face. This strategy is related to the concept of Z-ordering in computer graphics. The reason to do this more complicated kind of drawing instead of a wireframe is that it gives a more realistic look to your virtual object – the box appears "solid" because it occludes the scene behind it.

Be aware that, depending on the structure of your particular 3D scene, any scene features IN FRONT of the virtual box from a particular camera view will not appear correct – they will look like the box is in front of them, hiding them from view, instead of occluding the box as they would if it had been a real object in the scene. It is OK if this happens to you, because it is a very hard problem to solve without segmenting out surfaces in the scene and computing their depths. Even "professional" SDKs like ARKit for iOS do NOT currently attempt to do this.

What to hand in

1) Submit your code, input images, output text files produced by COLMAP, and a pdf project report, all together in a single zip file on Canvas.
2) The project report should be a step-by-step description of how you achieved each of the 10 steps above, including explanations of how you solved each of the tasks, pointers to what part of the code solves each task (or you can include snippets of code in the report), present any equations you had to derive (e.g. how to fit a plane to a set of points; what are the projection equations you are implementing); how you went about setting up a local coordinate system associated with the dominant scene plane and the transformation between local coordinates and scene coordinates. Basically, whatever effort went into solving any of the tasks should be documented.
3) Include LOTS of figures showing intermediate steps, with captions saying what they are. Show the 3D point cloud produced by COLMAP. Show the inliers of the dominant plane you detected. Can you find a nice way to show the 2D local coordinate system that you defined with respect to the dominant plane (as an example, perhaps drawing 3D line segments representing the 2D x-y grid in the dominant plane)? Show your virtual box object together with the 3D scene point cloud. Of course show the final result, which is the 3D virtual box object projected into each image and rendered on top of the original pixel values, ideally as a "solid-looking" box that occludes the pixels behind it. The more images of intermediate and final results you show in the report, the better, not just to make it visually please (although that is nice) but also because the tasks to be solved are very geometric and lend themselves well to visual debugging – looking at pictures or interactively rotated 3D plots will immediately tell you (and me) if you are getting the expected results for each step.
4) Finally, for multi-person groups, include a statement of what each person did to contribute to the project. It is expected that not everyone will work on every task, there of course will be some division of labor. But try to ensure that everyone contributes in roughly equal amounts.