Stereoscopic Depth Mapping Team

Eric Henderson, Nick Kamper, Laura Moss, James Savage

Project Plan, Timeline, and Preliminary Research

# Project Plan

We plan to split the project into three phases – determining and implementing the best point-matching algorithm, implementing the stereo depth mapping algorithm, and determining and implementing 3D object face detection. If time permits, we would like to implement a 3D rendering of the scene as determined from the stereo depth information and the object face detection.

If time does not permit, we will choose to only implement a point-matching algorithm and depth-mapping algorithm, as we know that both of these can be implemented in a reasonable amount of time. Both of these algorithms are fairly well studied and should be easy to implement within the timeframe given.

# Timeline

End of Week 1 – Determine, Implement, and Benchmark Point-Matching Algorithm. Prepare test set of image data.

End of Week 2 – Implement Stereo Depth Mapping Algorithm

End of Week 3 – Begin to implement 3D object face detection and prepare for demonstration and class presentation

# Research

Since the literature review, we have setup a Git repository for our code, images, and documentation. We have also discussed options for getting a stereo camera apparatus setup, but we have decided to just use publically available stereo images for the mean time until we can build such a rig and equip it with cameras. We found a good resource of coplanar stereoscopic images with known camera characteristics through the Birmingham Object Lighting Database (BOLD), which is located at <http://bold.bham.ac.uk/find_external.php?light&page=1&ipp=20>. These images give us a significantly large corpus of images to work with. Note that we can only use the external BOLD images, as the other sets of images do have the cameras pointed toe-in (therefore, they aren’t coplanar). In the future, we could handle toe-in images, but that would require performing a de-skewing operation to map the images back to the same backplane, which would be outside of the scope of this class.

Figure – Rig Design

In addition, we found another set of stereoscopic images at <http://vision.middlebury.edu/stereo/data/>, which can also be used to augment our data set and check the performance of our algorithms.

After doing some preliminary research as part of the literature review, we believe that one of our best hopes for a point-matching algorithm is one of the HMATCH or C2MODEL/LMATCH algorithms described in the SRI paper. We will still need to experiment to see if the rough camera parameters from C2MODEL are accurate enough for LMATCH to work sufficiently, which will be done once we have an actual implementation. In addition, we will attempt to manually calibrate a camera and get the camera parameters to compare them with the parameters derived by C2MODEL.

Once we have a few of the “seed” matches, provided either by HMATCH or LMATCH, we will then use an algorithm similar to GMATCH to create even more matching pairs of points. Once we have a sufficient number of matching points (preferably, approximately half of the pixels in the image, at best), we can then determine the depth of those points from the camera. In the coming days, we plan to actually begin implementing HMATCH, C2MODEL/LMATCH, and GMATCH algorithms in MATLAB and begin performing tests to see if they perform as expected. If not, we will continue to find other point matching algorithms that may or may not be more general-purpose than the SRI algorithms, which were designed for stereoscopic imaging.

For the depth-mapping portion of the project, we know the simple trig for calculating the depth in terms of pixels, and we can map from pixel size to physical size. This places the stipulation that we know the field of view of the camera in addition to the distance of the cameras from one another.

One of the potential areas of research would be algorithmically handling pairs of points that are labeled as being the same point without being the same physical point. Our final algorithm should be fairly tolerant of this type of “noise.”