



16-682 - MRSD Project II | ILR #10
Individual Lab Report #10 | April 6, 2017

SAMBUDDHA SARKAR

Team G
eXcalibR

Huan-Yang Chang
Man-ning Chen
Sambuddha Sarkar
Siddharth Raina
Yiqing Cai

1. INDIVIDUAL PROGRESS

1.1 Overview

In this ILR I have tried to show tangible visuals or results for the work I have been doing. As of now, the Image generation pipeline with data import/export capabilities is ready and ahead of schedule.

The virtual environment is being modeled in an open source platform: Blender 3D 7.68a. It is a Maya based platform and is programmable by Python 3.

Topics covered have been listed below for a quick overview.

- 1.2 Visualizing the planes of patterns
- 1.3 Importing data in Blender (Camera, Trajectory, Object)
- 1.4 Automated Image Generation
- 1.5 Blender Render Pipeline Status

1.2 Visualizing the planes of the patterns

The process of marking the geometric locations of the face patterns involves the assignment of a local reference frame on the calibration target (any one vertex of the target), then assigning each face of the target a local reference frame and finally relating all these frames to the world frame. The reference frame for each individual face is set according to some parameters relating to the geometry of the calibration target itself. The Z axis is aligned along the face normal, the X-Y axes are chosen such that each face's geometric pattern can be described using just one single description file. The description file output (with custom .dsc, .calib extensions) was explained in ILR#09. In figure 1.2.1 One can see a scene in blender where only the calibration target. Now in figure 1.2.2 one can visualize the reference frames of the faces as well as the calibration target with the script that I have written. This script stores the required data for exporting inside the blender environment and its data blocks, this means that everything is contained inside a very small sized blender file (.blend extension) and can be generated on demand on any computer or terminal.

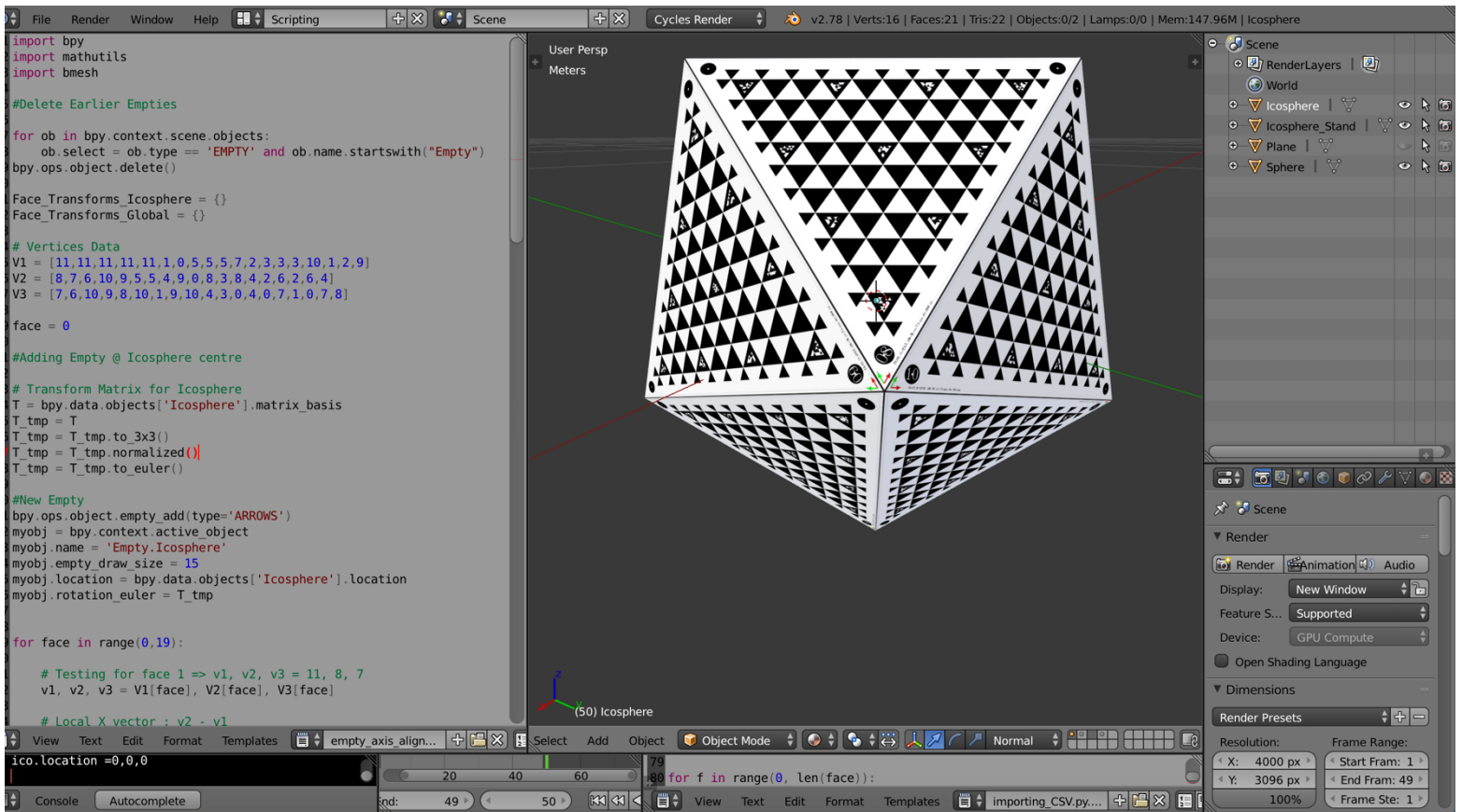
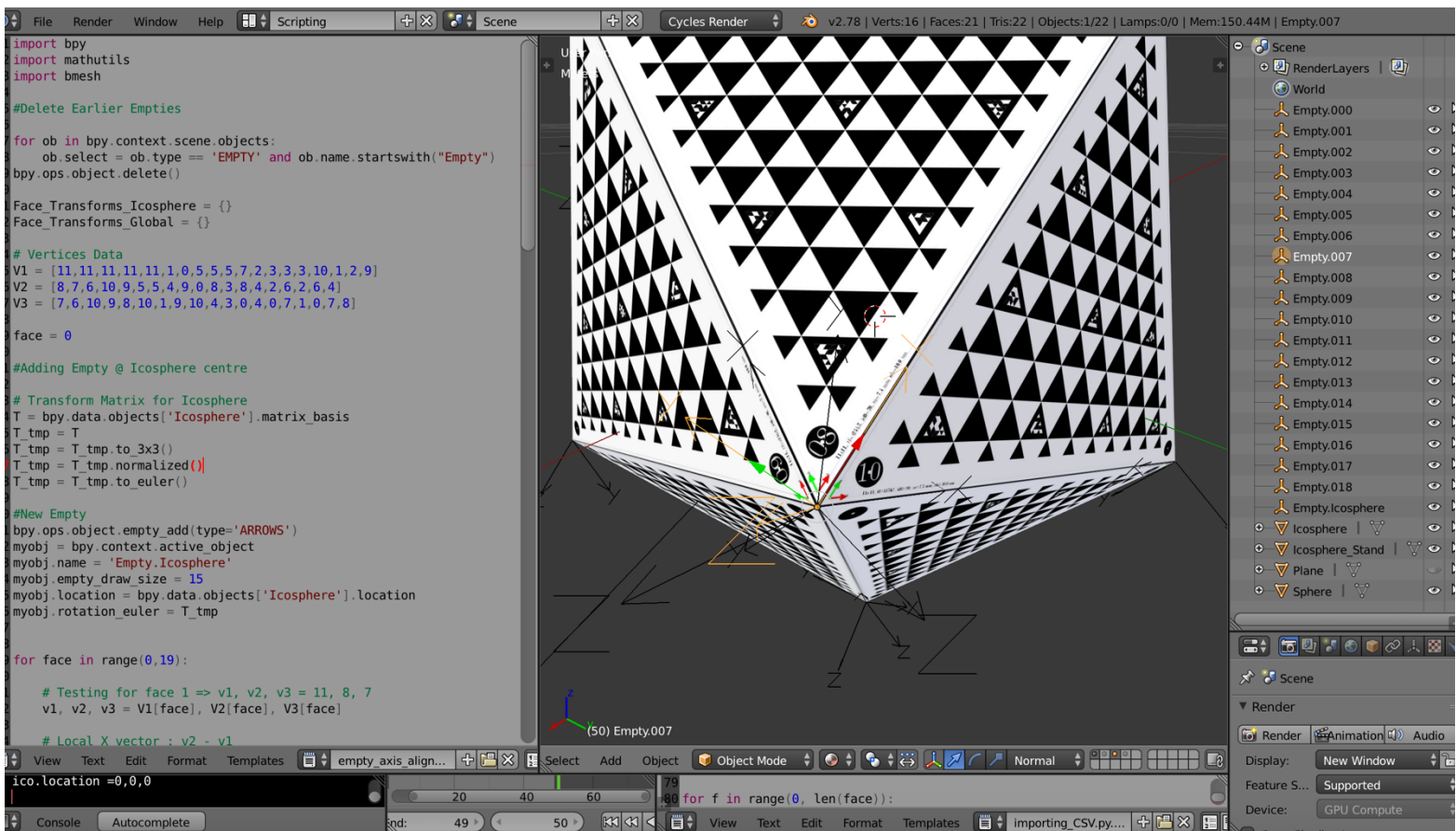


Fig. 1.2.1, Calibration Target without mappings.



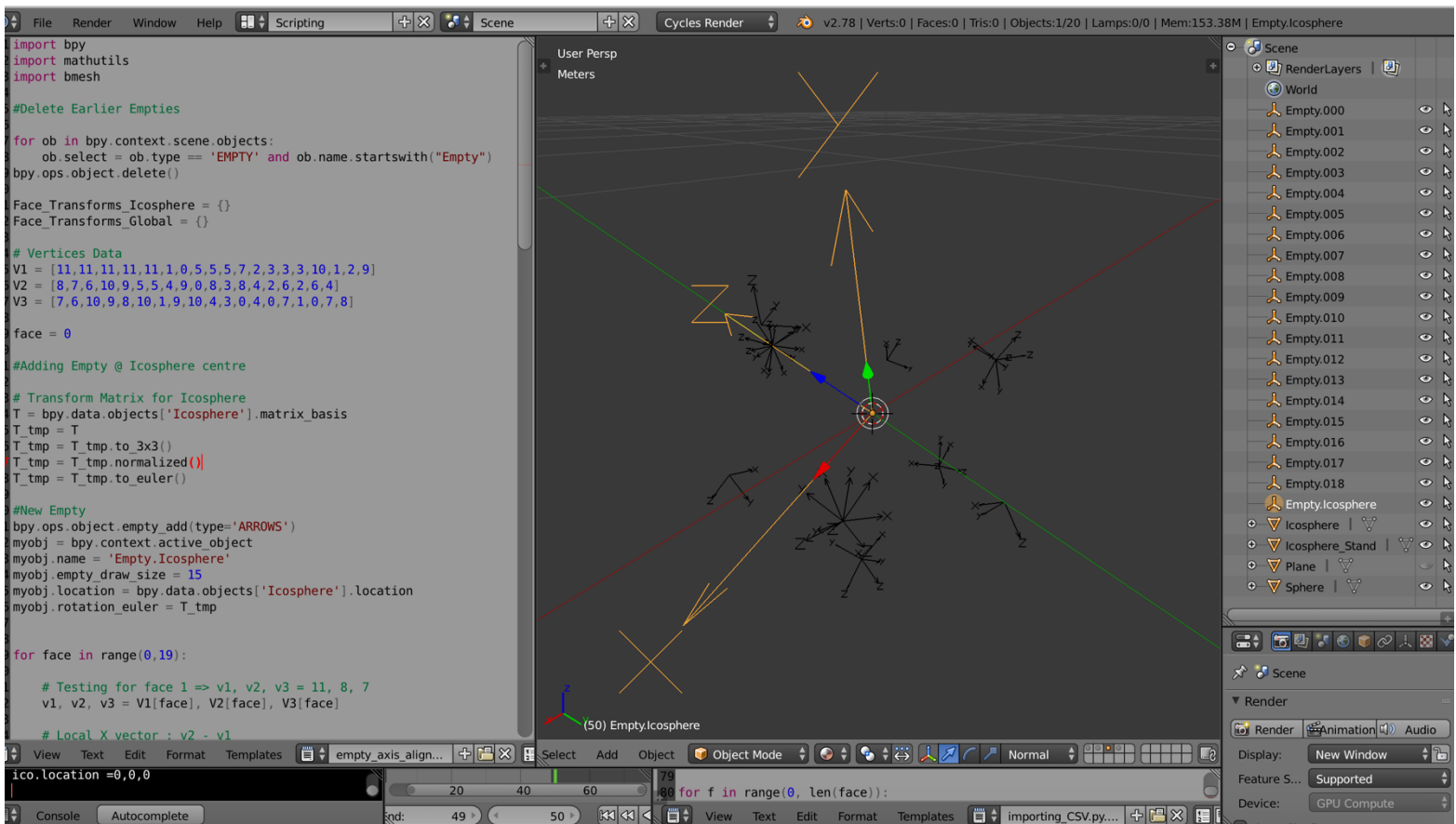


Fig. 1.2.2, Mappings on Calibration Target Visualized (with & without target)

In figure 1.2.2 one can clearly see the axes of different faces in a right-hand coordinate description. The left-hand side of the image gives a peek into the script for axes data and map data generation & storing. The right-hand side of the image depicts the coordinate transform data stored in the blender environment named in an ordered manner. To clarify 'Empty.000' stored the data for face id = 000 and so on, whereas 'Empty.Icosphere' stores data for the whole target.

1.3 Importing data in Blender (Camera, Trajectory, Object)

The Camera Data and Trajectory Data for the objects are imported into blender using a predefined CSV. I have just the screen shots of different stages as I explained the mechanics in the blender pipeline section.

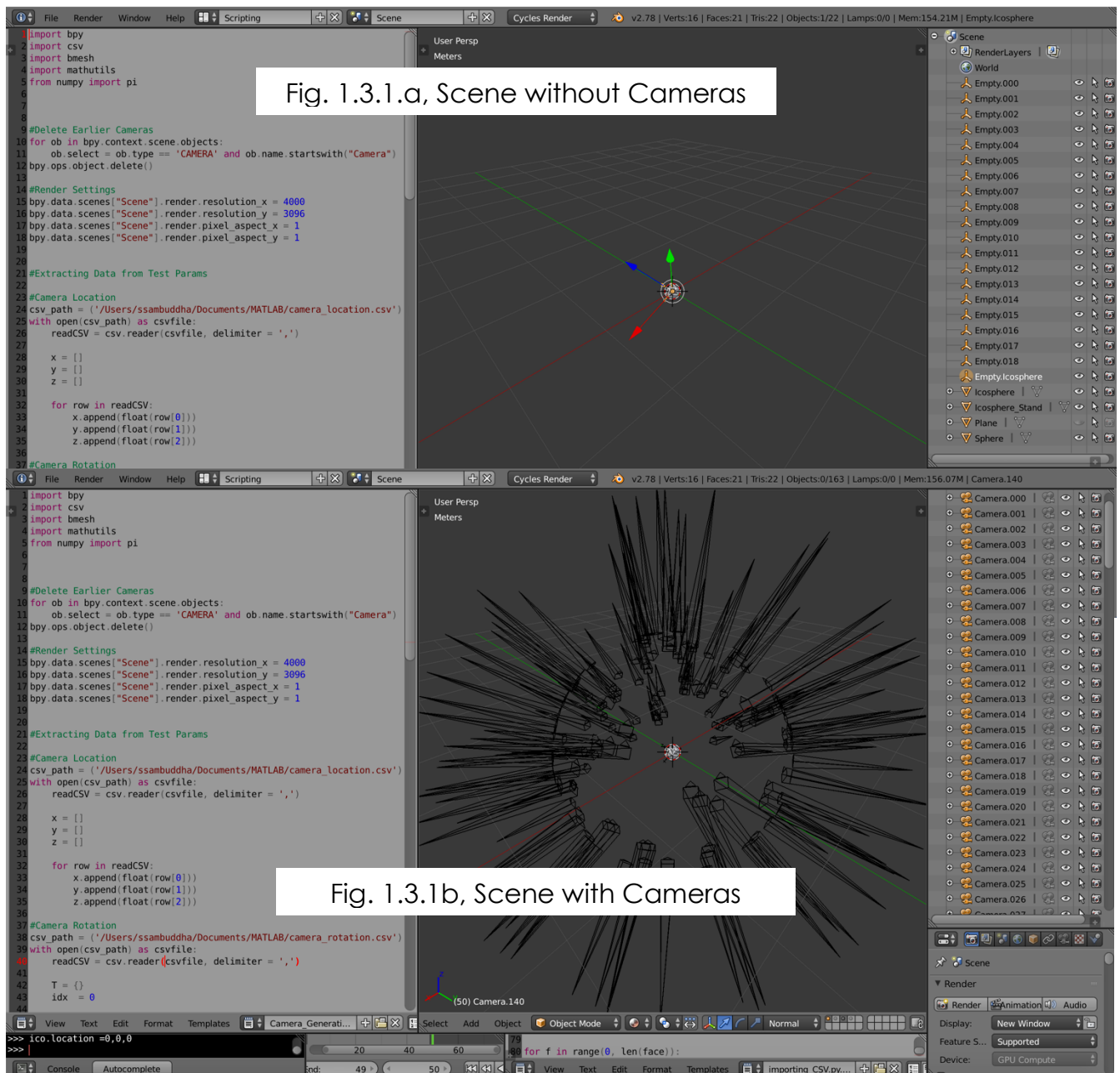
The figures follow this convention:

Left-hand side :Script

Center: Output/ Visualization

Right-hand side: Data blocks relating to the generated objects

1.3.1 Camera Generation: Fig. 1.3.1.a and 1.3.1.b



1.3.2 Trajectory Data: Fig. 1.3.2.a and 1.3.2.b (The yellow lines represent keyframe insertion i.e. animation data for the trajectory and trigger points for the camera to capture images.)

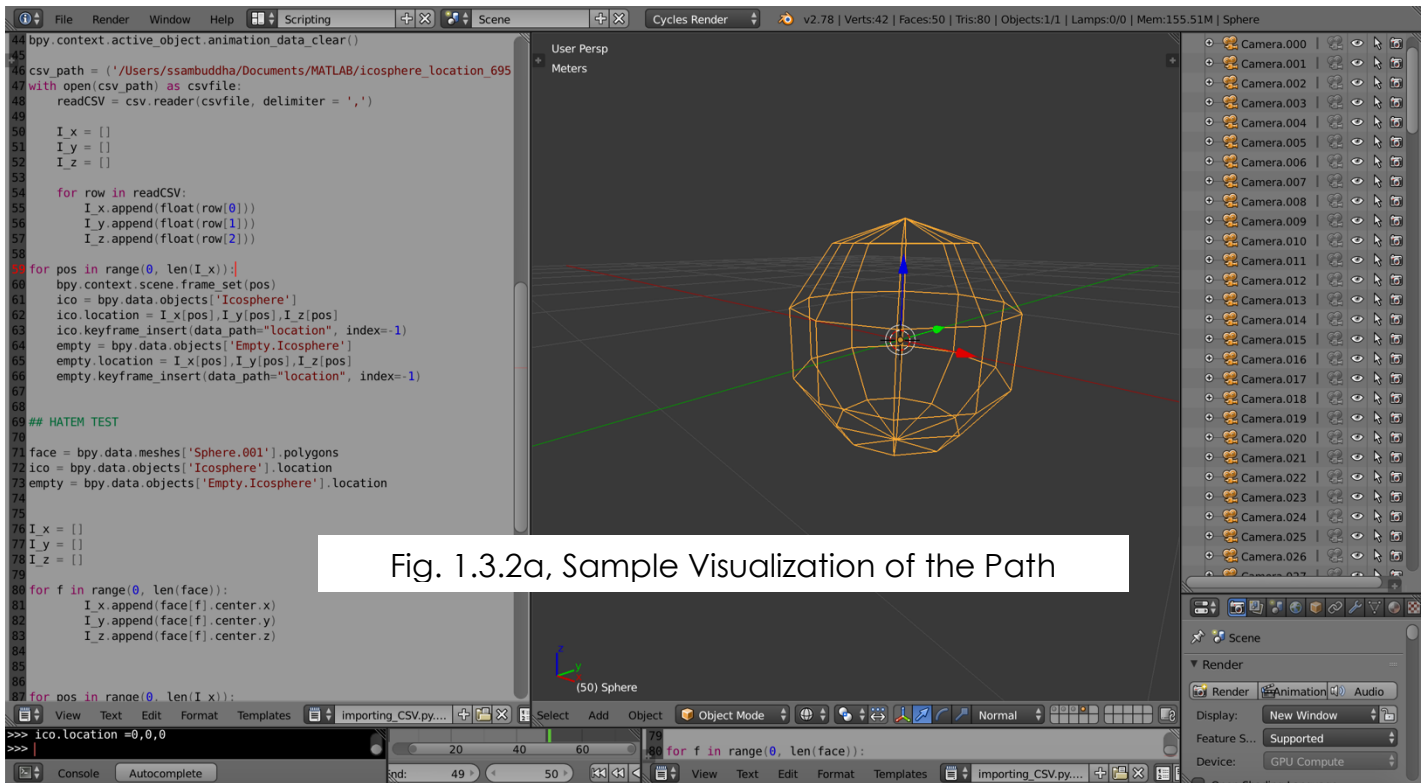


Fig. 1.3.2a, Sample Visualization of the Path

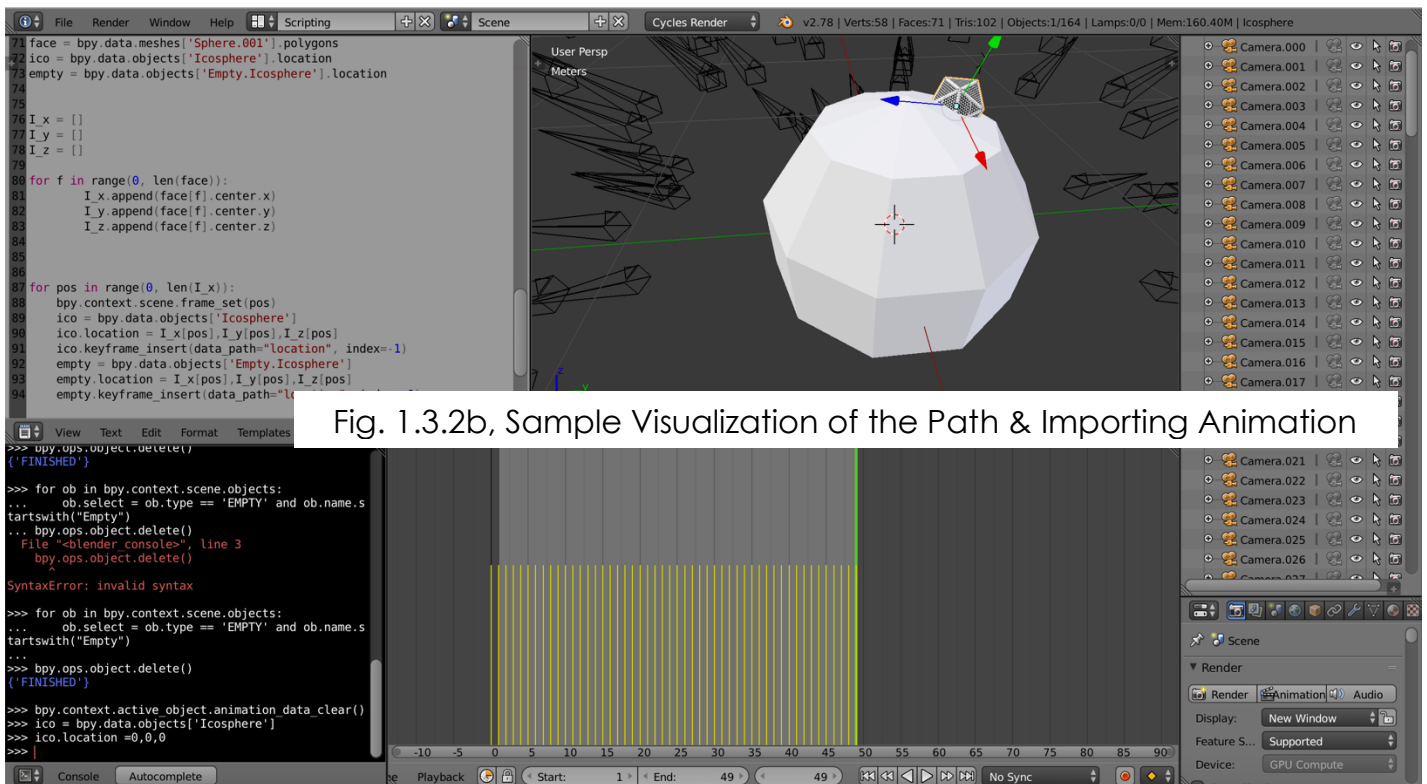
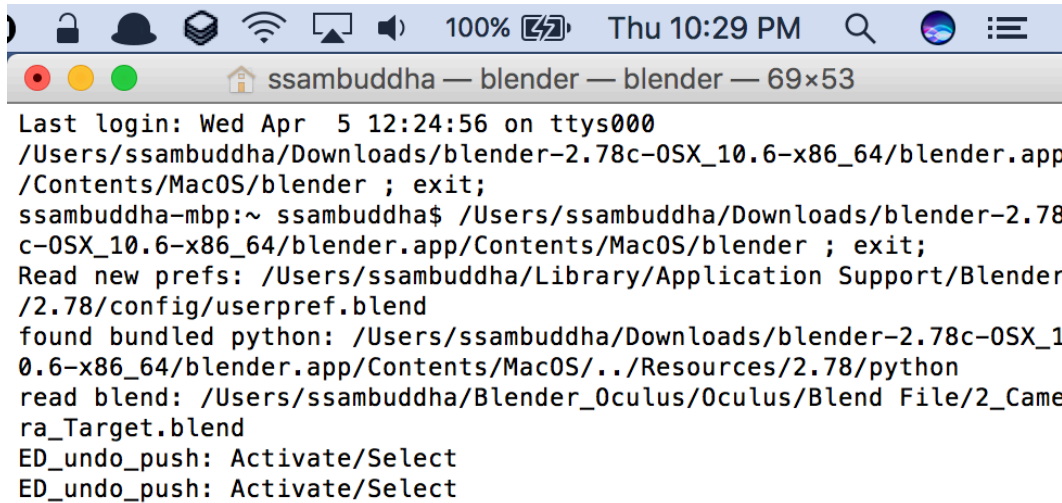


Fig. 1.3.2b, Sample Visualization of the Path & Importing Animation

1.4 Automated Image Generation

The rendering for all the cameras can be pushed into the render stack by using a click of a button (as demonstrated in the PR#11). I have included a screen shot of such a test a performed while writing this ILR. Figures from 1.4.1 to 1.4.3 show the different stages of the live test. I used 3 cameras with terrible render settings just to demonstrate that the pipeline is actually functional. The output is shown via a diagnostic snippet I wrote to monitor the status of the render pipeline. This diagnostic tool can be activated in the blender environment on MAC/Windows/Linux with ease is compatible with Terminal/Command Prompt.



```

Last login: Wed Apr 5 12:24:56 on ttys000
/Users/ssambuddha/Downloads/blender-2.78c-OSX_10.6-x86_64/blender.app
/Contents/MacOS/blender ; exit;
ssambuddha-mbp:~ ssambuddha$ /Users/ssambuddha/Downloads/blender-2.78
c-OSX_10.6-x86_64/blender.app/Contents/MacOS/blender ; exit;
Read new prefs: /Users/ssambuddha/Library/Application Support/Blender
/2.78/config/userpref.blend
found bundled python: /Users/ssambuddha/Downloads/blender-2.78c-OSX_1
0.6-x86_64/blender.app/Contents/MacOS/./Resources/2.78/python
read blend: /Users/ssambuddha/Blender_Oculus/Oculus/Blend File/2_Came
ra_Target.blend
ED_undo_push: Activate/Select
_ED_undo_push: Activate/Select
```

Fig. 1.4.1, Start of the render process (note the time at the top)

```
eval fcurve 'location' - 47.000000 => 47/50, 1
eval fcurve 'location' - 47.000000 => 47/50, 1
Saved: '/Users/ssambuddha/Blender_Oculus/Oculus/Live_ILR10_Test/Camera.002/47.png'
Time: 00:05.87 (Saving: 00:00.93)

Evaluate all animation - 47.000000
eval fcurve 'location' - 47.000000 => 47/50, 1
eval fcurve 'location' - 47.000000 => 47/50, 1
eval fcurve 'location' - 47.000000 => 47/50, 1
eval fcurve 'location' - 47.000000 => 47/50, 1
eval fcurve 'location' - 47.000000 => 47/50, 1
eval fcurve 'location' - 47.000000 => 47/50, 1
Evaluate all animation - 48.000000
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
Evaluate all animation - 48.000000
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
Saved: '/Users/ssambuddha/Blender_Oculus/Oculus/Live_ILR10_Test/Camera.002/48.png'
Time: 00:04.66 (Saving: 00:00.65)

Evaluate all animation - 48.000000
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
eval fcurve 'location' - 48.000000 => 48/50, 1
Evaluate all animation - 49.000000
Evaluate all animation - 49.000000
Saved: '/Users/ssambuddha/Blender_Oculus/Oculus/Live_ILR10_Test/Camera.002/49.png'
Time: 00:04.49 (Saving: 00:00.61)

Evaluate all animation - 49.000000
ED_undo_push: Run Script
2017-04-06 22:41:37.234 blender[77211:2522340] IMKClient Stall detected, *please Report* your user scenario attaching a spindump (or sysdiagnose) that captures the problem - (imkxpc_presentFunctionRowItemTextInputViewWithEndpoint:reply:) block performed very slowly (672.28 secs).
2017-04-06 22:41:37.238 blender[77211:2522340] IMKClient Stall detected, *please Report* your user scenario attaching a spindump (or sysdi
```

Fig. 1.4.2, End of the render process (note the time at the top)

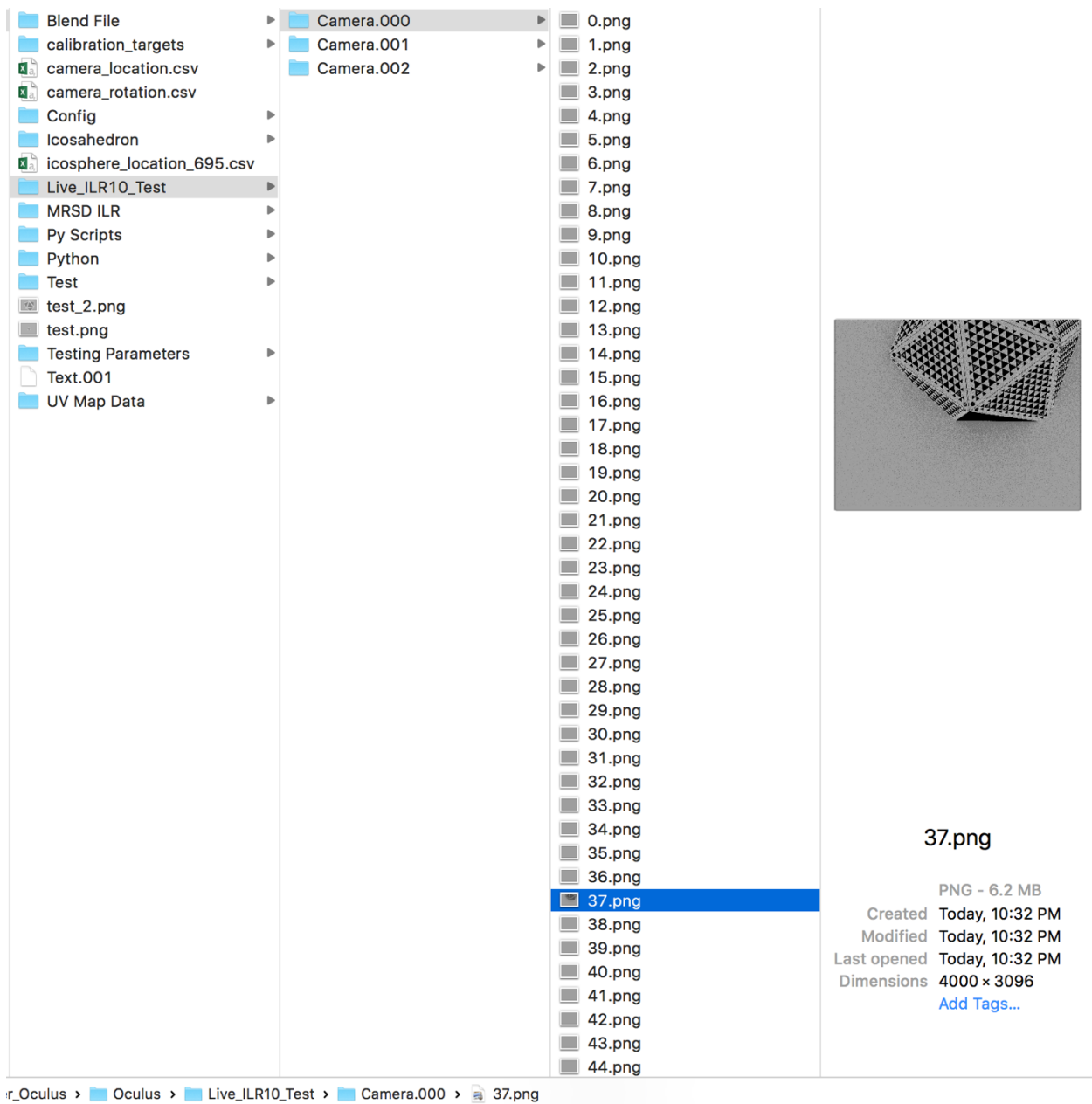


Fig. 1.4.3, Final output tagged according to camera and keyframe number.

1.5 Blender Render pipeline

The blender rendering pipeline (Fig 1.5.1) developed to aide semi-autonomous generation of image data sets for the geometric calibration algorithm. The calibration pipeline illustrated is self-explanatory. The

pipeline loads in camera data (intrinsic & extrinsic), object data (calibration target), required configuration and the number of the cameras and the render settings. It spews out Images and other mesh data using the cycles render engine and bpy module. The current progress, that is all the modules for the pipeline have been completed is shown in Fig. 1.5.2,

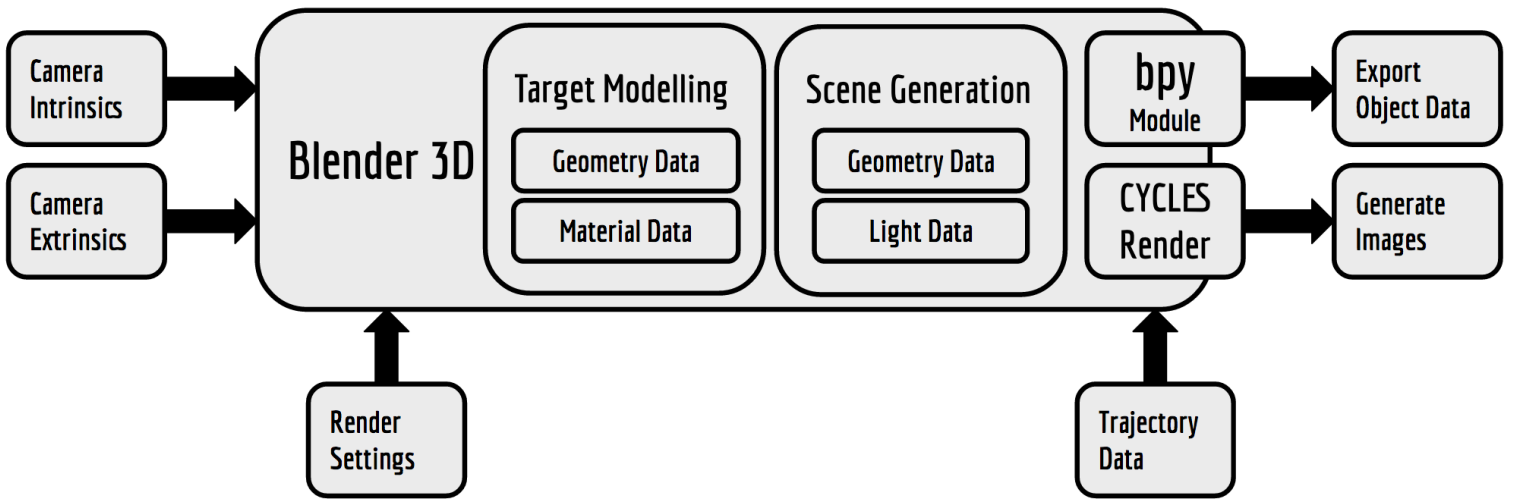


Fig. 1.5.1, Overall Pipeline.

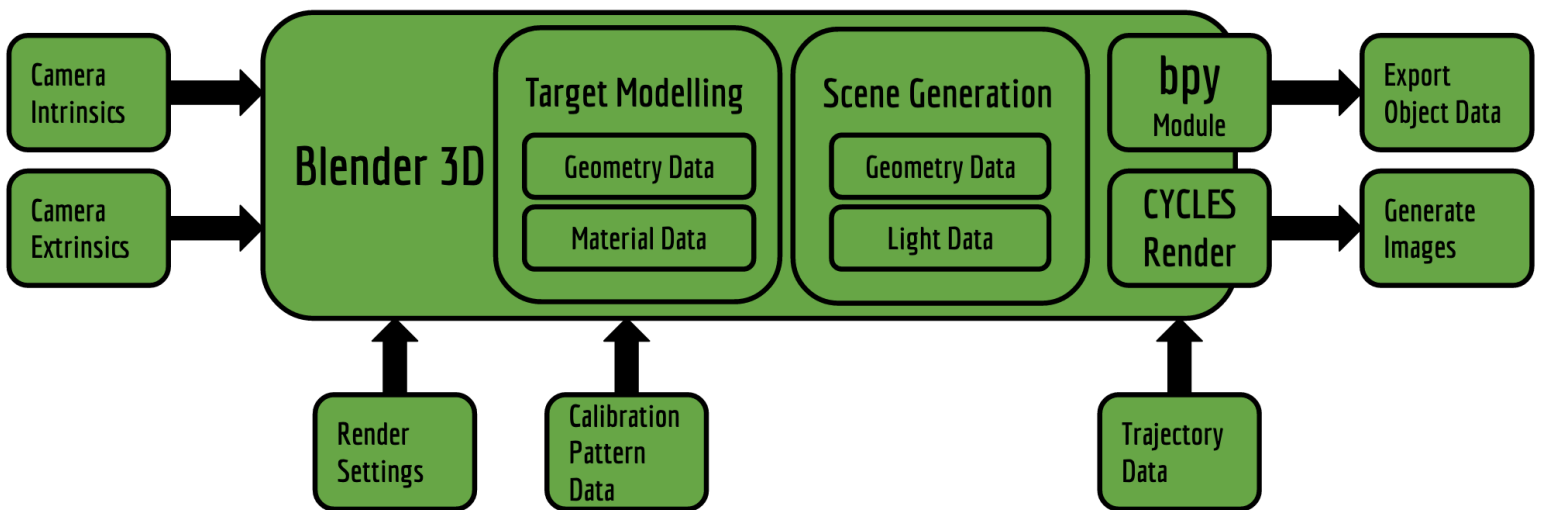


Fig. 1.5.2, Current progress on the pipeline.

2. CHALLENGES

The main challenge right now is rendering images from all 141 cameras in our virtual test environment. There are about 700 images per camera per scene, so in one scene one image per camera takes up a time of 1 min on an average (averaged over a set of 100 images). So, for all 700 images on each of the 141 cameras to render, it would take approximately two months. This estimate is based on the current GPU configuration at Oculus, so the render time can be cut short a lot just by using render farms i.e. online servers dedicated to rendering outputs for blender projects. The discussion related logistics are underway with Oculus as the blender files do contain sensitive data pertaining to the calibration pattern.

3. TEAM WORK

The project work was divided among the team members and the task was assigned according to the strengths of the team members. The task division has been listed below in Table 3. The divided tasks can be completed in parallel; hence others can pitch in when some team members fall behind in their work.

Team Member	Task
Huan-Yang Chang	ABB robot operation in real-time to capture images.
Siddarth Raina	Sensor Noise.
Man-nig Chen	Color Calibration: Mapping function.
Yiqing Cai	Multiple Camera Model: Validating Scoring heuristics for the original path.
Sambuddha Sarkar	Blender Render pipeline and Geometric Calibration tests.

Table 3, Task Division.

4. FUTURE PLANS

My future plans until the next progress report is to reduce the render time for the image generation and test geometric calibration results with different image data sets generated from Blender 3D.