Using Synthetic 3D Images in Designing Algorithm

Zili Wang

March 2016

1 Introduction

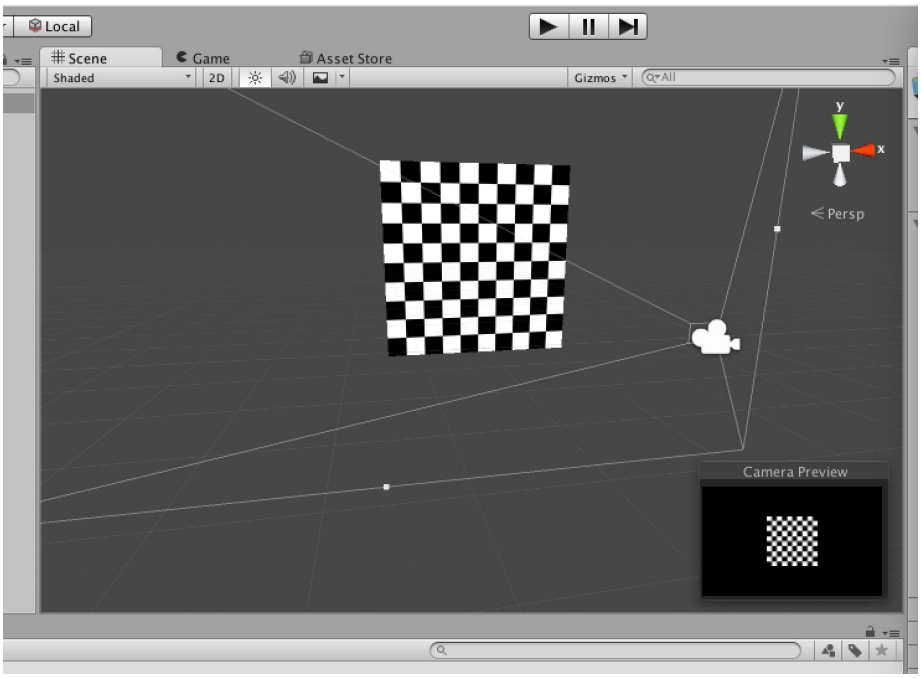
Using 3d rendering technique, we can generate the virtual 3d scene. Applying CV algorithm on these synthetic images has several advantages:

1. We know exactly the ground truth
2. We can get rid of the noise we don't want, and add the noise we expected.

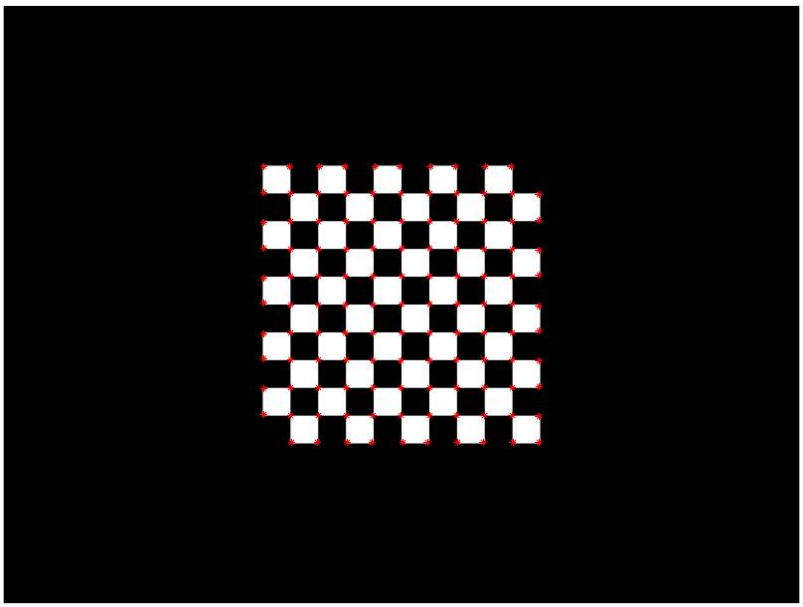
With these, it is very useful in testing and studying the algorithm. So, in this work, I implement two example of using Synthetic 3D Images. For the 3d rendering tool, I used Unity3d. Of cause I can use openGL, but Unity3d is more flexible, I can easily use high level function to implement UI, interactive operation, and also I can manipulate all the low level details.

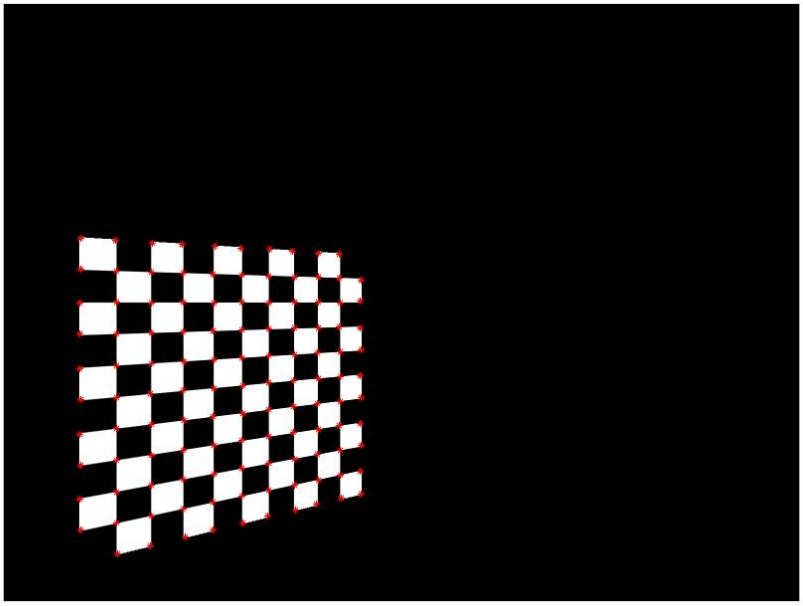
2 Calibration Pattern

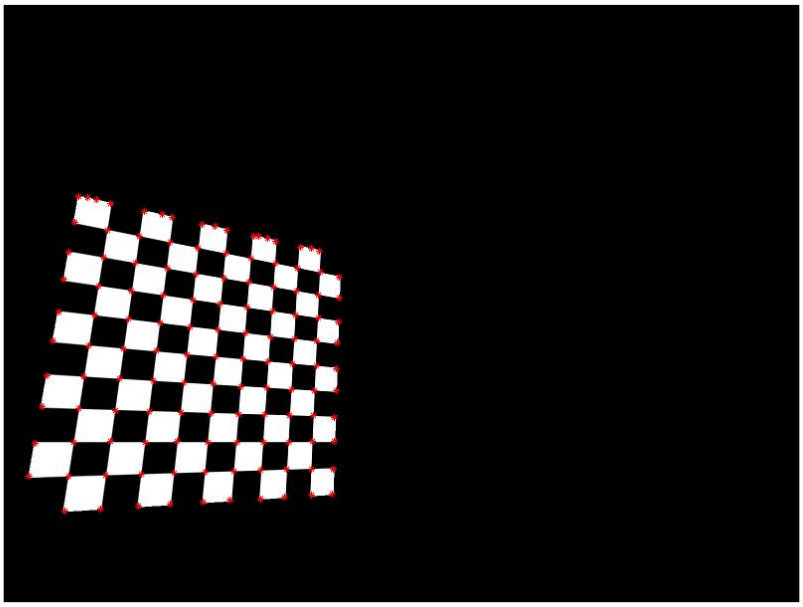
I apply a checkerboard texture to a plane. By moving the camera I can get different images with different view direction. Then I use Harris corner detector to extract corners.

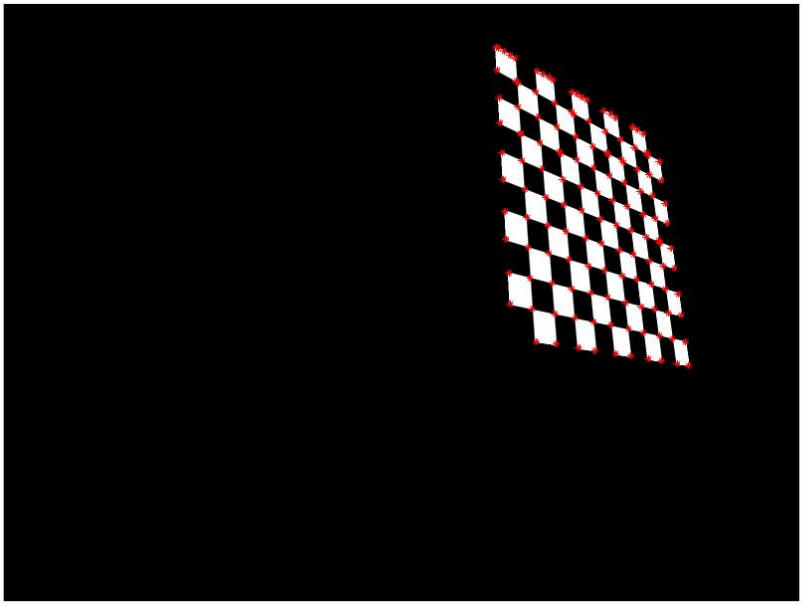


Seeing the checkerboard plane in Unity3d









We can know the 3d position of each corner very easily, because we know the position of the plane.

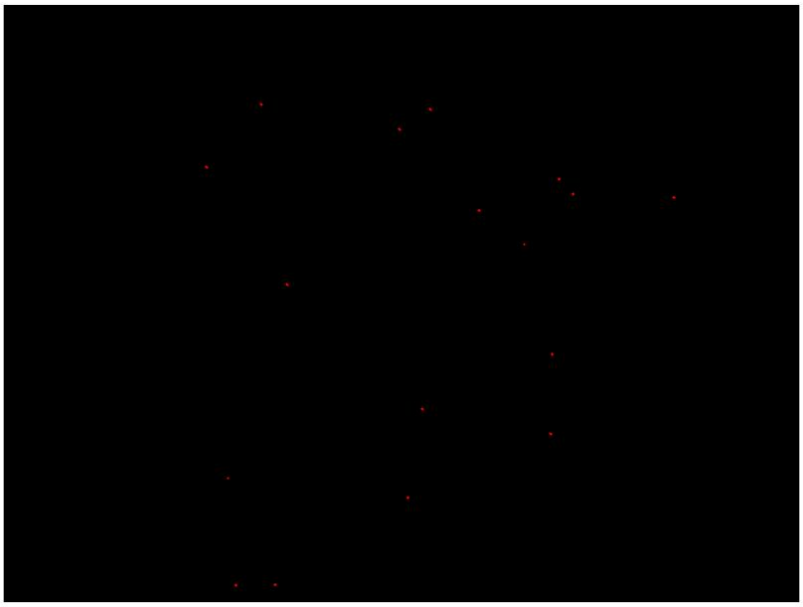
3 Corresponding 3d Points with 2d Points

I used a special rendering mode to just render the discrete points. And I use the value of the point to code the index of a point.
Detection of points in images is easy, since we can get rid of almost all the noise. So I just simply test the value of each pixel. And calculate the index of point based the value of pixel. So I know which pixel on image correspond to which point in 3d space.
But there are still two kinds of noise we need to consider:

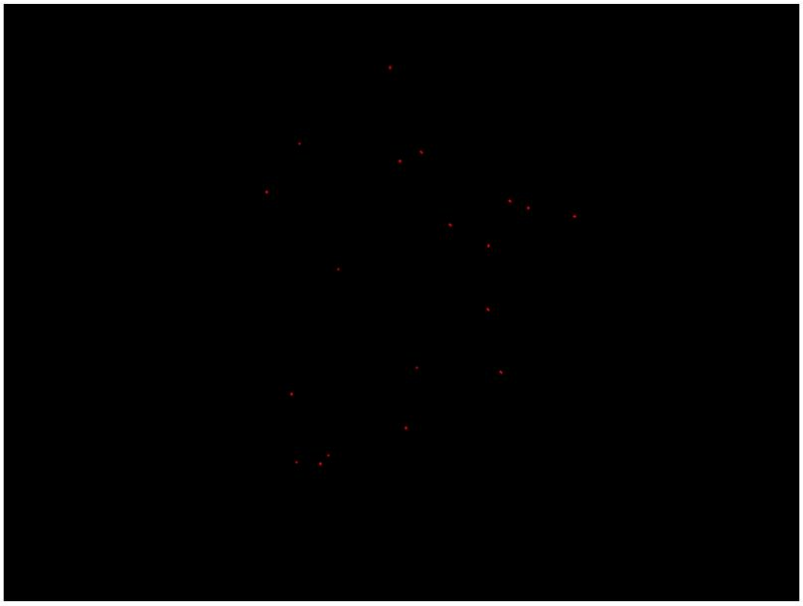
1. Some points in 3d can project to more than one pixels on the image.
2. In the whole process, the value of a point or pixel is converted between char and float multiple times.

I can either use algorithm to filter out these noise, or I can manipulate some very low level features in rendering pipeline to solve these problems.

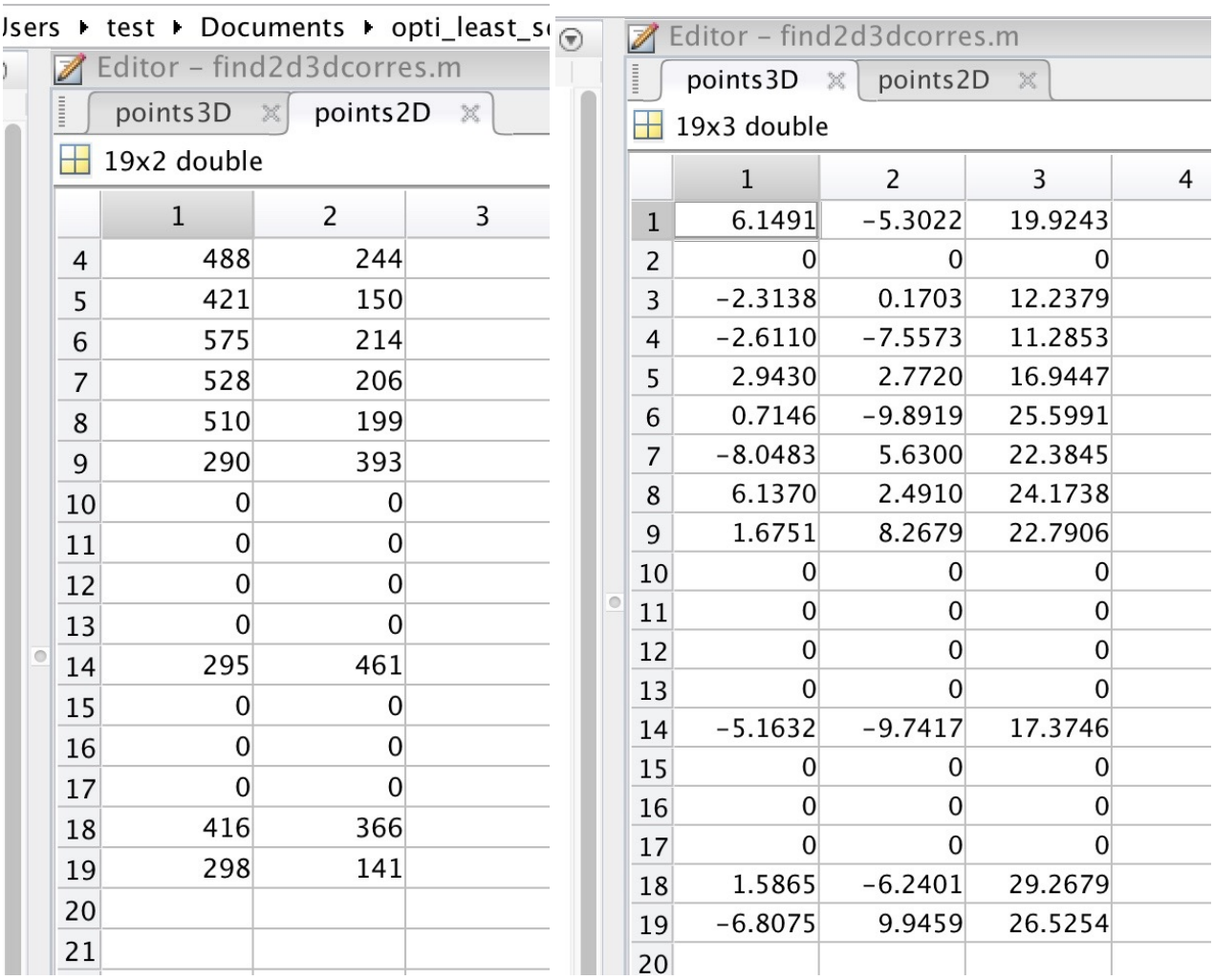
For some points, there are two pixel corresponding to it. These two pixels have the same value. We can using the average position of these two pixels.
The intensity of each point code the information of the point index.



These are images seeing the same point cloud from different direction.



I export the 3d position of each point to csv file, and import it in matlab. In matlab, I connect each 2d feature to its 3d position.



	1	2	3
4	488	244	
5	421	150	
6	575	214	
7	528	206	
8	510	199	
9	290	393	
10	0	0	
11	0	0	
12	0	0	
13	0	0	
14	295	461	
15	0	0	
16	0	0	
17	0	0	
18	416	366	
19	298	141	
20			
21			

	1	2	3	4
1	6.1491	-5.3022	19.9243	
2	0	0	0	
3	-2.3138	0.1703	12.2379	
4	-2.6110	-7.5573	11.2853	
5	2.9430	2.7720	16.9447	
6	0.7146	-9.8919	25.5991	
7	-8.0483	5.6300	22.3845	
8	6.1370	2.4910	24.1738	
9	1.6751	8.2679	22.7906	
10	0	0	0	
11	0	0	0	
12	0	0	0	
13	0	0	0	
14	-5.1632	-9.7417	17.3746	
15	0	0	0	
16	0	0	0	
17	0	0	0	
18	1.5865	-6.2401	29.2679	
19	-6.8075	9.9459	26.5254	
20				

This is the resulting correspondence, for those entries that are 0, mean that these points do not appear in this image. Then I can use these information to calculate the pose of my camera.

4 Codes

All the related codes are on github: https://github.com/ziliwangmoe/synt_image_cv