MineNav: An Expandable Synthetic Dataset Based On Minecraft For Aircraft Visual Navigation

We present a novel synthetic dataset MinNav based on the sandbox game Minecraft. This dataset uses several plug-in program to generate rendered image sequences with time-aligned depth maps, surface normal maps and camera poses. MinNav is a highly scalable dataset, users not only can easily obtain a large number of big-scale 3d scene files in the community, saving modeling time, but also can build specific scenes in the game. what's more, thanks for its open source feature, users can also develop modules to obtain other ground-truth for different purpose in research. Different from other synthetic datasets, our proposed dataset has a community of a large number of players, which can build 3d scenes and obtain tools at a lower cost, and because there are a large number of light and shadow rendering tools, the generated synthetic dataset can be greatly reduced Distribution deviation from real-world data.

Understanding scenes through video is a significant research in visual perception. It includes many classical computer vision tasks, such as depth recovery, surface normal prediction and visual odometry etc. Undoubtedly, data sets are the top priority of research.

Presently datasets[7, 4] has already been applied in industrial such as autonomous driving [3], interactive collaborative robotics [5], and localization and navigation systems [6], etc . But ground-truth of these always suffer the approximately measurement limited by the sensor, or even unavailable, requiring huge cost. Most synthetic dataset[2, 10] based on open-source film makes up for the above shortcomings partly. It also provide a new opportunity for computer vision research. What's more ,the data bias between synthetic data and real-world data is unavoidable problem and limited amount of data, especially the scene, is Gradually unable to meet demand of resent model.

We propose a simply method to generate high quality synthetic dataset based on open-source game Minecraft includes rendered image, Depth map, surface normal map, and 6-dof camera trajectory. This dataset has a perfect ground-truth generated by plug-in program, and thanks for the large game's community, there is an extremely large number of 3D open-world environment, users can find suitable scenes for shooting and build data sets through it and they can also build scenes in-game. as such, We don't need to worry about manual over fitting caused by too small datasets. what's more, there is also a shader community which We can use to minimize data bias between rendered images and real-images as little as possible. Last but not least, we now provide three tools to generate the data for depth prediction ,surface normal prediction and visual odometry, user can also develop the plug-in module for other vision task like segmentation or optical flow prediction.

2 Preparatory tools

It is a sandbox video game created by Swedish game developer Markus Persson and released by Mojang in 2011. The game allows players to build with a variety of different

blocks in a 3D procedurally generated world, and has already been a tools in several research[9, 1, 11]. It minimum component is block sized 1×1×11111\times 1\times 11 × 1 × 1, and map loading range is a square with 1536×1536153615361536\times 15361536 × 1536 , support player to redevelop plug-in module to achieve specific function.

Replay Mod

is a Modification for the Minecraft which allows players to record and replay their gaming experience with monocular, stereo, or even 360D videos. Player can generate dynamic 3d scene file centered as main player and set camera trajectories manually with adjustable fov.The 3d scene file support rendered by Blender, and also could be rendered in real-time by third-party shader.

Optifine

is a Minecraft optimization mod. It allows Minecraft to run faster and look better with full support for HD textures and many configuration options. We have developed two shader3 thought it to generate precise ground-truth in sync with image sequences.

Sildur

Sildur is an open-source shader written in GLSL, it adds shadows, dynamic lighting, and waving grass, leaves and water to increase the reality , reduce the data bias between rendered data and real-data.

3 Generation of Datasets

In this paper, we choose a big game map AudiaCity 2.0 (Fig. 4) as scene to build MinNav.

It contains over 1,500 buildings , covering an area of 16 square kilometers and an altitude of 67 meters. including schools, hospitals, libraries,wharves and factories etc, has a good diversity to meet most demand111https://www.planetminecraft.com/project/audia-project-minecraft-city/. \dirtree.1 MinNav. .2 Grids Number. .3 Trajectory Number. .4 color. .5 frame-number.png. .5 …. .4 depth. .5 frame-number.png. .5 …. .4 timestamp.txt. .4 camera-state.txt. \dirtree.1 Grids Raw Files. .2 grid-number. .3 SceneNumber.mcpr. .3 timelines.json. .3 …. .2 ….

It obviously that the map can not loaded into limited memory all at once, We sampling every 400 meters in both directions divided the whole map into several Grids sized as a 800m×800800800m\times 800800 italic_m × 800 and saved them as dynamic 3d scene by ReplayMod (Sec. 2) . For each grid, we manually set 1 to 3 camera trajectory(ies) over 20ms, and totally generating 168 grid directory including 8800 samples as MinNav which includes color image, depth map, surface normal image and 6-dof camera trajectory.

Data Description

The raw recording data such as dynamic 3d scene and camera trajectories of each sample-grid are zipped by replay mod into a single file named SceneNumber.mpcr. Besides the raw recording, we also provide post-precess data(MinNav sequences) i.e., rectified and synchronized video frames, where the GridNumber and TrajectoryNumber are placeholders for sampled-grid number and it's camera trajectory number. For data synchronized, we start from 0ms and generate a set of rendered images(color, depth and normal) and camera 6-dof poses every 100ms through raw recording data.

Image

There are about 200 to 300 monocular color images rendered by Sildur for each camera trajectory, and stored with loss-less compression using 8-bit PNG file. Same with other synthetic dataset, We can rendered MinNav at any spatial resolution, and the fps can also adjust from 10 to 120, here we render at 800x600 with fov=70 and fps=10. The ReplayMod also support other Image format such as stereo or 360d, which takes possibilities for other vision applications6.

Depth

The depthshader written in GLSL support by Optifine. We get the precisely depth of each block in view and map into [0,1]01[0,1][0 , 1], rendering them as gray-scale map stored with 8bit-depth PNG file(Fig.1). For the rendering speed, the gray-scale and depth are not completely linear, so we rectify and restored the depth as npy file to make sure the depth in game and gray of stored pixels are in a 1: 1 linear relationship. The max map loading area are about 2.56 Square kilometers center as player, The block out of range would render as 1 gray denotes over far.

Surface Normal Color Encoding

We also write Surface Normal Shader by GLSL to infer the surface normal of scene in view. The world coordinates (The coords system see Fig.7) has three base vectorz-,x-,y+superscriptsuperscriptsuperscriptz^-,x^-,y^+italic_z start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT , italic_x start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT , italic_y start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT , camera rotation has tree dof Pitch,yaw,roll. We mapped the 6 directions surfcaces into BGR color values. . if pitch=,yaw=,roll=formulae-sequenceformulae-sequencepitch=\alpha,yaw=\beta,roll=\gammaitalic_p italic_i italic_t italic_c italic_h = italic_ , italic_y italic_a italic_w = italic_ , italic_r italic_o italic_l italic_l = italic_ then:

Cx-=(coscos,sin,1)subscriptsuperscript1\displaystyle C_x^-=(\cos\beta\cos\gamma,\sin\gamma,1)italic_C start_POSTSUBSCRIPT italic_x start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_cos italic_ roman_cos italic_ , roman_sin italic_ , 1)

Cx+=(1,1,sin)subscriptsuperscript11\displaystyle C_x^+=(1,1,\sin\beta)italic_C start_POSTSUBSCRIPT italic_x start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , 1 , roman_sin italic_)

Cy+=(1,coscos,sin)subscriptsuperscript1\displaystyle C_y^+=(1,\cos\alpha\cos\gamma,\sin\alpha)italic_C start_POSTSUBSCRIPT italic_y start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , roman_cos italic_ roman_cos italic_ , roman_sin italic_)

Cy-=(sin,1,1)subscriptsuperscript11\displaystyle C_y^-=(\sin\gamma,1,1)italic_C start_POSTSUBSCRIPT italic_y start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_sin italic_ , 1 , 1)

Cz-=(sin,1,coscos)subscriptsuperscript1\displaystyle C_z^-=(\sin\beta,1,\cos\alpha\cos\beta)italic_C start_POSTSUBSCRIPT italic_z start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_sin italic_ , 1 , roman_cos italic_ roman_cos italic_)

Cz+=(1,sin,1)subscriptsuperscript11\displaystyle C_z^+=(1,\sin\alpha,1)italic_C start_POSTSUBSCRIPT italic_z start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , roman_sin italic_ , 1)
Cx-,Cy+,Cz-subscriptsuperscriptsubscriptsuperscriptsubscriptsuperscriptC_x^-,C_y^+,C_z^-italic_C start_POSTSUBSCRIPT italic_x start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT , italic_C start_POSTSUBSCRIPT italic_y start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT , italic_C start_POSTSUBSCRIPT italic_z start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT denotes the BGR value of a Surface normal in the same direction as the base vector x-,y+,z-superscriptsuperscriptsuperscriptx^-,y^+,z^-italic_x start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT , italic_y start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT , italic_z start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT respectively, See appendixA for color mapping of single degree of freedom rotation.

Camera Pose

For each grid, we manually set 1 to 3 camera trajectories, taking videos within map loading area. We set a keyframes contained camera 6-dof pose such as x,y,z,pitch,yaw,roll every 5 seconds in replay mod which, using CatmullRom with =0.5,n=50formulae-sequence0.550\alpha=0.5,n=50italic_ = 0.5 , italic_n = 50 to fill in the data points between two key frames. Due to the timelinse.json only recorded keyframe information, we coding to achieve CatmullRom externally to get camera pose for each 100ms.

We sampled the camera pose by ourself tools by extracting the grid raw files3, the result are shown in 8.

4 Domain Randomization

Using domain randomization[12] is a popular method for introducing more diversity to a synthetic datasets. Some works generate images with different render passes.[2] generates each frame that simulate different aspects of image formation such as smooth shading, specular reflections, and inter-reflections. Although some [10] rendered their self-made scenes in multiple versions, each with random incremental changes to the cameraâs translation and rotation keyframes. transformations enrich the datasets , they still are similarity preserving.

In addition to generating different scenes to increase the diversity of dataset such as deserts, snow, jungles, lakes, cities, etc., MC also has a time system and a weather system for achieving this (Fig.9). All of them can change the brightness and visibility of a dynamic scene, expanding diversity of dataset in a better way. Every sample-grid can apply to different weather or time for achieving reduction of a network’s generalization error.What’s more, using different shader for one grid-sample, or adjusting the fog options or motion blur in Replay Mod can also increase the diversity.

5 Experiment

The experiments is designed to investigate how MinNav useful is for generalizing to real-world aerial image dataset like VisDrone[14], and shows the running result in serveral popular alogrithms.

Evaluating the Deviation of data distribution

In this section, we seek to validating the Deviation of data distribution between MinNav and VisDrone compared with kitti[7].

We originally planned to pre-train in MinNav and KITTI respectively through the same model, and then evaluate it in visdrone, confirm which data set is better for generalization on visdrone. However, Due to the absecnt of ground-truth in VisDrone, We changed our approatch. We train an unsupervised model in visdrone, and then quantify the evaluation on MinNav and kitti respectively, and get the quantitative results(See tab.1). The better evaluation results on MinNav means that the data distribution of this dataset is closer to MinNav.

It can be seen that the same model is directly evaluated on MinNav after being trained by KITTI and VisDrone respectively.The latter result is significantly better than the former.It can be seen that although the MinNav data set is a synthetic dataset, it is more than KITTI in data distribution. In order to be close to VisDrone, so for the data model based on aerial images, MinNav is better than kitti for depth estimation tasks.

Evaluation Results on Recent Works

In this section we trained a depth estimation and odometry estimation model monodepth2[8] on our dataset and the quantitative result are shown in Tab.2

This paper provided a simply method to generate synthetic dataset with ground-truth of depth , surface normal and camera trajectory. Basic texture and geometry information given by Minecraft is obviously not enough for a dataset to training model, while through the high quality shader provided by community, the color image can narrow the data distribution gap as much as possible. What's more By developing module in Minecraft, user can easily obtain needed ground-truth for other computer vision task such as optical flow, segmentation label with a low cost. Traditional synthetic[2, 10] datasets are generated by open source movie and Rendered by professional 3d render software like Blender. thought with a lower threshold of building dataset, it still need users with a tedious work, especially for the large scale scene building. Thanks for large community, The are Massive maps can be easily found on the website and users never worry about the shortage. We further found that recent depth prediction algorithms that perform well on the KITTI benchmark do significantly worse on the MinCV data set, suggesting room for new methods.

There are several ways users can expand our current dataset, These include rendering the color image at higher spatial and temporal resolutions, with different shader or add motion blur, with additional game map, or setting render options as stereo or cubic.

Appendix A Normal Vector Color Maping

In this section we mainly show the color mapping of surface by changing pitch, yaw and roll angle independently. if pitch =absent=\alpha= italic_, then:

Sy+=(1,cos,sin)subscriptsuperscript1\displaystyle S_y^+=(1,\cos\alpha,\sin\alpha)italic_S start_POSTSUBSCRIPT italic_y start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , roman_cos italic_ , roman_sin italic_)

Sz-=(1,1,cos)subscriptsuperscript11\displaystyle S_z^-=(1,1,\cos\alpha)italic_S start_POSTSUBSCRIPT italic_z start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , 1 , roman_cos italic_)

Sz+=(1,sin,1)subscriptsuperscript11\displaystyle S_z^+=(1,\sin\alpha,1)italic_S start_POSTSUBSCRIPT italic_z start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , roman_sin italic_ , 1)

similarly, if yaw =absent=\beta= italic_ or roll =absent=\gamma= italic_ then:

Sz-=(sin,1,cos)subscriptsuperscript1\displaystyle S_z^-=(\sin\beta,1,\cos\beta)italic_S start_POSTSUBSCRIPT italic_z start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_sin italic_ , 1 , roman_cos italic_)

Sx-=(cos,1,1)subscriptsuperscript11\displaystyle S_x^-=(\cos\beta,1,1)italic_S start_POSTSUBSCRIPT italic_x start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_cos italic_ , 1 , 1)

Sx+=(1,1,sin)subscriptsuperscript11\displaystyle S_x^+=(1,1,\sin\beta)italic_S start_POSTSUBSCRIPT italic_x start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , 1 , roman_sin italic_)

Sx-=(cos,sin,1)subscriptsuperscript1\displaystyle S_x^-=(\cos\gamma,\sin\gamma,1)italic_S start_POSTSUBSCRIPT italic_x start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_cos italic_ , roman_sin italic_ , 1)

Sy+=(1,cos,1)subscriptsuperscript11\displaystyle S_y^+=(1,\cos\gamma,1)italic_S start_POSTSUBSCRIPT italic_y start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (1 , roman_cos italic_ , 1)

Sy-=(sin,1,1)subscriptsuperscript11\displaystyle S_y^-=(\sin\gamma,1,1)italic_S start_POSTSUBSCRIPT italic_y start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_POSTSUBSCRIPT = (roman_sin italic_ , 1 , 1)