Unity based Urban Environment Simulation for Autonomous Vehicle Stereo Vision Evaluation

Matija Vukić, Borna Grgić, Dorian Dinčir, Luka Kostelac, Ivan Marković
University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{matija.vukic, borna.grgic, dorian.dincir, luka.kostelac, ivan.markovic}@fer.hr

***Abstract*—Testing and evaluation of sensor processing algorithms for autonomous vehicles is challenging due to the problem of collecting reference data. To ensure safety and robustness, many man-hours need to be spent in collecting and preparing such data. One solution to alleviate this problem is to use computer simulations. Computer simulations can model a real system with all its static and dynamic characteristics. This approach provides efficiency and precision when collecting data and reduces testing time. The aim of this paper is development of a simulation environment based on Unity where it would be possible to test sensors and algorithms for autonomous vehicles and show deviations from reference data. The proposed simulation model contains typical city objects and participants: roads, sidewalks, buildings, pedestrians, traffic signs and vehicles. In this paper we simulate motion and sensors from a single vehicle equipped with a stereo camera setup. The program environment Unity is used for designing the simulation, and behavioral scripts are executed with C# programming language. To showcase the testing of applicable algorithms, OpenCV class for computing stereo correspondence, using the semi-global block matching algorithm, is used on simulated stereo images and we discuss future development of the simulation.**

***Index Terms*—Autonomous Vehicles, Urban Scenario Simulation, Stereo Vision**

I. Introduction

Reliable perception is one of the key components of any autonomous system and entails analyzing and understanding a series of steps; from physical properties of the operation principle by which sensor collects data to high level semantic reasoning about the scene. Combining different sensor modalities enhances the chances of an autonomous robot or vehicle operating robustly in a wide range of scenarios. However, collecting sensor data in real-life scenarios takes resources and time; thus, in order to alleviate that step and prepare more adequately, prior testing and analysis can be performed in appropriately designed simulated environments. Indeed, recent reinforcement learning research extensively leverages simulations in order to train policies that would take impractically long in real-life and transfer them to reality [1].

Another challenge, besides collecting the real-world datasets, is getting reference, i.e., ground truth, data for evaluation purposes. For example, testing accuracy of vehicle localization based on stereo cameras or 3D laser range finders can be compared to more accurate sensor such as a differential GPS as done in the well-known KITTI dataset [2]. Another KITTI dateset example is evaluation of disparity computation from stereo cameras by comparing it to the point cloud measured by the 3D laser range sensor; however, depth evaluation for every stereo camera frame is not available since depth integration from several 3D scans is necessary. Furthermore, once datasets are recorded, evaluating semantic classification and multitarget tracking algorithms requires annotations by human experts [3]. Naturally, real-life datasets and experiments constitute a fundamental block and are irreplaceable for the development of autonomous vehicle algorithms and valuable insight papers have reported on the experience and lessons learned [4]–[6]; nevertheless, insights, testing, and reduction of development costs can be achieved by also exploiting benefits of appropriately designed simulation environments.

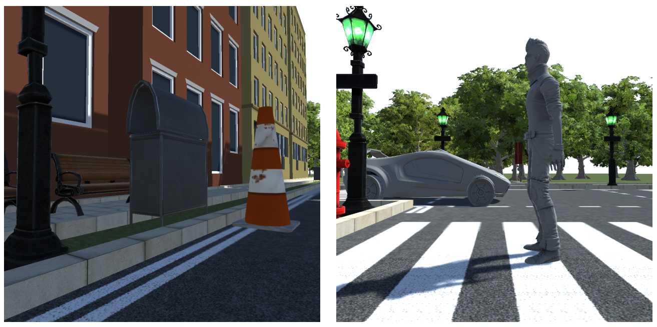


Fig. 1: Left camera images of the generated street curb and crossroad examples

In [7] a virtual urban environment was presented for conducting experiments with a fleet of autonomous vehicles as part of the project aiming to design novel transportation systems based on a fleet of small electric cars supervised by a central computer. An approach to urban traffic scenario simulator implementation and pertaining requirements along with an overview comparing at the time available simulators and the possibility of simulating DARPA Urban Challenge teams sensors were discussed in [8]. A lightweight simulator aiming at replicating urban features, such as road networks, curbs and general objects, including a realistic host vehicle was proposed in [9]. Therein, for the purposes of motion planning the authors also modeled other traffic participants which could interact with each other relying on intelligent agent-based approaches. In [10] a photo-realistic virtual version of the KITTI dataset was presented that was automatically labeled with accurate ground truth for object detection, tracking, scene segmentation, depth and optical flow. The created dataset was use to demonstrate that deep learning algorithms pretrained on real data behave similarly in the real-world and the simulated one, while pre-training on virtual data improves performance. Research in [11] presented the development of an urban driving simulator named CARLA. It is an open-source simulator for development, training and validation of urban autonomous driving systems and offers flexible specification of sensor suites and environmental conditions. In the paper, authors also compared the performance of three different approaches to autonomous driving. Furthermore, in [12] used CARLA for analyzing the challenge of transferring driving policies learned in simulated environments to the real world. Given the above discussed advantages of simulated urban environments for testing development of autonomous vehicles, there exists multiple open source and commercial simulators [13]–[19].

NVIDIA Drive Constellation [14] uses the computing power of two different servers to create a cloud-based computing platform for autonomous vehicle testing. The first server runs the DRIVE Sim software that uses GPUs to generate a wide range of testing environments and scenarios. The second server contains DRIVE AGX Pegasus AI car computer that processes the simulated data as if it were coming from the sensors of an actual car. DRIVE Sim positions virtual car in the environment and sends data from sensors to Pegasus. Pegasus processes the data and sends driving commands back to DRIVE Sim. Together, the two servers create a digital feedback loop. Cognata [16] uses computer vision and deep learning algorithms to automatically generate a whole virtual city environment. Different weather conditions and lighting are added to stress test the system. Simulation engine combines TrueLife and PhysicsStudio to simulate the sensor interaction with the external materials to receive the most comprehensive autonomous driving simulation feedback loop. Virtual simulator enables to run thousands of different scenarios based on various geographic locations and driver behaviors. Apollo Simulation [15] is one of the solutions of the Apollo3.5 Platform. Its open functionality allows users to input different road types, obstacles, driving plans and traffic light states. Likewise, execution modes give users complete setup to run multiple scenarios and verify uploaded modules in the Apollo environment. Integrated Automatic Grading System in simulations tests via ten metrics; some of which are: speed limit, collision detection, traffic light recognition etc. It also provides users with 3D visualization of real-time road conditions and visualizes module output while showing the status of the autonomous vehicle. Cvedias SynCity [17] simulates various environments for autonomous applications, also providing sensor simulations of a vast number of real-world phenomena such as weather, day/night cycle and different traffic situations. It has flexible API that gives users the ability to customize before mentioned environment and independently control parameter and the scenario. It automatically generates ground truth for users with various occlusions and visibility constraints.

In the present paper we construct a simulated urban environment based on the freely available Unity engine. We model an urban-like neighborhood including other agents, such as cars and pedestrians, including intersections with traffic lights. The present paper focuses on the perception side of the autonomous vehicle control challenge; concretely, stereo vision. Stereo cameras are an omnipresent and popular sensor setup for autonomous vehicles, since they enables accurate ego-motion estimation and simultaneous localization and mapping, while also, being a vision sensor, they can be directly used for other vision-based challenges, such as semantic scene interpretation. The simulated vehicle with a stereo camera can drive along any predefined path and record data (examples of left camera images are shown in Fig. 1). Besides the two cameras simulating a stereo sensor setup, we placed a third camera that uses a special shader, so called Z-buffer, to generate ground truth depth information. Given that, disparity estimation algorithm accuracy can be compared to this reference data having at all times also available ground truth motion of the vehicle itself. We present an example of stereo disparity estimation from simulated stereo images using the semi-global matching algorithm [20].

The paper is organized as follows. In Section II we present the Unity 3D framework and describe some of its capabilities relating to simulating environments for autonomous vehicles. In Section III we describe the static and dynamic parts of the constructed urban environment, while in Section IV stereo image capture as well as ground truth generation is explained. In the end, Section V concludes the paper.

II. Unity 3D Simulation Framework

Unity is a cross-platform game engine developed by Unity Technologies and can be used for three-dimensional and two-dimensional games as well as simulations. It was first released in 2005 as an OS X-exclusive game engine. Over the years, Unity become available on 27 platforms; from standard computer operating systems and popular consoles to virtual and mixed reality devices. In the sequel we describe some of the used Unity features and tools [21].

A. Lighting

Lighting is provided by light objects or by creating ambient light and emissive materials. Unity has two basic lighting techniques: realtime and precomputed. In some situations both techniques can be combined to create a more realistic lighting scene. Directional, spot and point lights are set to realtime by default. Realtime lighting is basic way of lighting objects in the scene, where light rays from realtime lights do not bounce from the surface of object. In order to make a more realistic scenes we need to use global illumination that is part of the precomputed lighting technique. Static lighting effects are calculated and then stored in a reference texture map called a lightmap and this process is named *baking*. The effects of light sources on static objects in the scene are calculated and stored to textures. Static lightmaps cannot react to changes in lighting conditions but precomputed realtime global illumination offers us a technique for updating scene lighting interactively. Lights can cast shadows from an object onto itself or onto other objects in scene. Shadows, naturally, add a degree of realism to the scene, because they bring out the scale and position of objects. Furthermore, for vision algorithms they can pose an additional challenge or be switched off for simplicity.

B. Cameras

Cameras in Unity are objects that transform a three-dimensional scene to a two-dimensional one which can be reproduced to viewer's screen. Position of camera defines the viewpoint and other components define the size and shape of the region that will be reproduced to viewer. A camera in the real world simulates perspective projection and this effect is for creating a realistic image. A camera that does not change the size of objects with distance is known as orthographic. Unity supports both views of the scene and they are known as camera projections. Perpendicular plane is set to the cameras forward direction to define the limit to how far camera can see. It is called the clipping plane, because objects at greater distance from the camera are clipped. There is also a corresponding near clipping plane that defines distance from camera at which objects will not be seen.

C. Mesh Geometry

Solid objects are made of a group of triangles arranged in three-dimensional space and is called a mesh. The mesh class stores all vertices in a single array, where each vertex is stored just once and each triangle is specified using three indexes of the vertex array. For correct shading the normal vector must be supplied for each vertex. A normal vector is perpendicular to the mesh surface at the position it is associated with. During shading each normal is compared with the direction of incoming light, thus when the two vectors are aligned the surface receives full brightness, otherwise it will be somewhere in between full brightness and complete dark.

D. Shader

Unity supports a wide range of shader types. Its features are enabled by using various texture slots and parameters. The standard shader incorporates *Physically Based Shading*. Physically Based Shading simulates the interactions between materials and light and has only recently become possible in real-time graphics. Physically Based Shading has a number of useful concepts, some of them are energy conservation and High Dynamic Range. Energy conservation ensures that objects never reflect more light than they receive.

E. Textures

Textures are standard bitmap images that are applied on the mesh to give better details since mesh only gives a rough approximation of the shape. For best representation of textures materials are applied. Materials use shaders to render texture on the surface of mesh, while shaders implement lighting and colouring effects to simulate different materials, because some materials are very complex they can combine multiple shaders. To ensure best representation of bumpy surfaces heightmaps are used. Heightmap stores an area where each point has a particular height from a baseline and are than converted to coordinates that are used to generate mesh.

F. Physics

Unity has implemented convincing physical behaviour that makes objects accelerate correctly and be affected by collisions and other forces. Two main components are rigid body and collider. When a rigid body is connected to an object it will immediately respond to gravity. Object with rigid body do not need rotation and position transformation to be moved around scene, instead forces should apply to push the object and let engine calculate the movements. Collider defines the shape of an object which responds in contact with other objects that have colliders applied to them. Collider is invisible and it approximates objects mesh to improve efficiency because mesh is usually complex object and calculations collisions between complex object slows down the physics engine.

III. Constructed Urban Simulation

The constructed simulation is placed in an urban environment implemented using the previously described Unity game engine. The environment contains typical city objects and participants like roads, sidewalks, buildings, pedestrians, traffic signs, and vehicles. To understand the implementation of these elements, Unity editor is briefly introduced. Most of the elements are constructed using standard assets provided by Unity or free assets from the Unity Asset Store.

A. Unity Editor interface

Unity provides a simple and user friendly interface. Two main tabs are Scene and Game. Scene tab shows current appearance of scene and enables numerous ways of editing added game objects and prefabs. All objects added in the scene are shown in Hierarchy tab where they can be arranged in desirable groups and order. Clicking on object in a scene or in the Hierarchy tab shows the objects Inspector tab. Inspector tab provides basic information about object and all components attached to it. Every object has Transform component that gives information about its local position, rotation and scale while wide range of other components (scripts, colliders and mashes, to name a few) is attached depending on objects usage. All imported assets and packages can be seen in Project tab and added to scene by simple drag and drop. Game tab, on the other hand, shows current game (or in this case, simulation) state. By clicking on play icon all components compile, notifying about potential errors and warnings in Console tab, and simulation starts from the main camera perspective.

B. Roads and sidewalks

Building of the proposed simulation starts by adding a Terrain game object on which sample scene of urban environment is placed. For the purpose of this simulation, terrain will be a

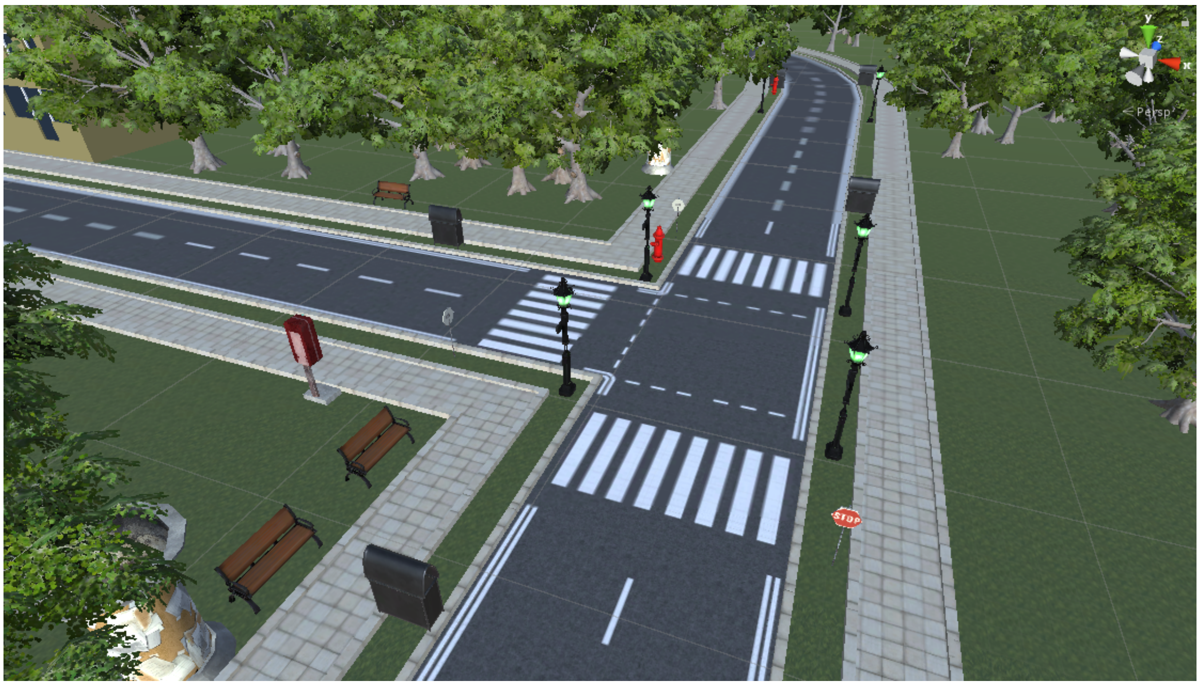


Fig. 2: Example of a generated intersection

big plain surface to simplify the rendering. To have a faithful representation of a city, it i necessary to add realistic roads and intersections. To achieve this, Simple Modular Street Kit and Low Poly Street Pack assets were used.These assets are available in Unity's Asset Store for free and enable various ways of implementing roads and intersections. Furthermore, the Low Poly Street Pack contains a large number of models of street signs and other street elements. By combining those two assets, making of city streets becomes a relatively fast process. The example of an intersection together with some other street elements is shown in Fig. 2.

C. *Buildings*

Besides roads, an integral part of every urban environment are its buildings. To surround the already implemented streets with them, the Simple Urban Buildings Pack 1, also free to import for Unity Asset Store, was utilized. It contains prefabs of five low poly buildings with already implemented optimisation technique called Level Of Detail (LOD) which reduces the load on the hardware and improves rendering performance when main camera is not close to them. Despite having only five buildings provided in asset, the city does not seem too repetitive thanks to diverse possibilities of modeling and rearranging the before mentioned prefabs.

D. *Vehicles and pedestrians*

Essential part of every urban environment simulation are agents, i.e., vehicles and pedestrians that populate it. To simulate these agents, available vehicles from Unity's Standard Assets were used. Standard Assets have everything needed for vehicles to move through streets by following defined way-points. CarWaypointBased prefab has an already implemented script components such as Car Controller, Car AI Control and Way-point Progress Tracker, that have numerous variables that can be changed to fine tune vehicle motion and physics. One of the vehicles is equipped with the main camera, whose perspective is seen when simulation is started, and two other cameras that simulate the stereo camera sensor. The main vehicle and game view are shown in Fig. 3.

E. *The final urban scene*

Using all the previously mentioned methods and elements, the implemented urban environment looks as shown in Fig. 4. The Terrains component for adding already existing trees in Unity was used to hide parts of it that should not be seen through main camera and to keep cities landlocked looks. Also, in Fig. 5 we shown the full Hierarchy tab with all the objects used to implement this example of an urban simulation.

IV. Capturing Stereo Images and Reference Data

Two cameras are used for capturing frames of the simulated city. They are mounted on the front of the car object and are symmetrical with respect to the car's main axis. These cameras are used to simulate the stereo camera with a fixed baseline and capture two images. Cameras were simulated with physical properties and had the focal length $f = 20\,\text{mm}$, baseline $b = 40\,\text{cm}$, and sensor size of $36\,\text{mm} \times 36\,\text{mm}$, while images were in ARGB32 color format and had resolution of 1024×1024. The quality of obtained disparity can then be checked by comparing it with the ground truth. Accompanying video showing an excerpt from the scene is available [1].

[1] https://youtu.be/2P0yYFxlVts

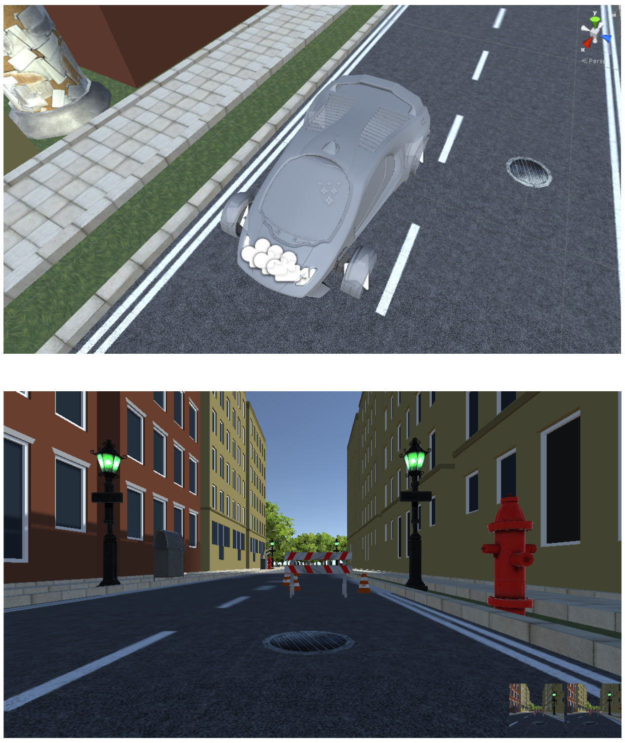


Fig. 3: Main vehicle and game view

A. *Constructing the disparity map and ground truth*

The semi-global block matching algorithm [20] is used to construct the disparity map from simulated stereo images. Implementation of this algorithm used in the paper is included in the OpenCV Library. The resulting image shows the distance of objects from the camera in a given frame. The distances are shown in the form of a gray-scale image, where parts of the disparity image that are closer to the camera are brighter and vice versa.

To compute the ground truth, a separate third camera was placed in between the stereo pair and used a special shader (Z-buffer) that is ran on the graphics card for every pixel in the frame. The shader encodes the depth in the form of grayscale images. Note that this discretizes the depth and for scenes with large depth variations various Z-buffer images should be created with different clipping planes as was done in the Tsukuba dataset [22]. Given the known depth Z, the corresponding ground truth disparity $d^\star$ for either left or right camera can be computed using the formula

$$d^\star = k \cdot \frac{fb}{Z}, \tag{1}$$

where k is the sensor scaling factor. Figures 6 and 7 show examples of the SGM computed disparity and the pertaining ground truth. We calculated the total error as percentage of erroneous disparities defined those that differ either absolutely by 5 px or relatively by 5% from the ground truth.

V. Conclusion

In this paper we presented a Unity based urban environment simulation for testing of autonomous vehicle perception algorithms. A model of a city was created together with other agents, such as cars and pedestrians, as well as intersections with traffic lights. A simulated vehicle equipped with sensors can then drive along a predefined trajectory and collect data. Specifically, in this paper we have demonstrated these capabilities for the omnipresent stereo camera by placing two virtual cameras on the moving vehicle. Besides capturing left and right camera images, the vehicle also has a dedicated third camera using a special shader that enables creation of depth ground truth data. Given that, we have shown how virtual stereo images can be used to compute the disparity image and compare it to the generated ground truth. For future work we plan to the city can be populated with more agents and other sensors can be added to the vehicle. Ground truth data can then be extended to 3D pointclouds and semantic labels.

References

[1] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. Mcgrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," in *arXiv:1808.00177*, 2018.

[2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[3] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015 : Towards a Benchmark for Multi-Target Tracking," *arXiv:1504.01942 [cs]*, 2015.

[4] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray, "Autonomous driving in urban environments: approaches , lessons and challenges," *Philosophical Transactions of the Royal Society*, vol. 368, pp. 4649–4672, 2010.

[5] M. Aeberhard, S. Rauch, M. Bahram, G. Tanzmeister, J. Thomas, Y. Pilat, F. Homm, W. Huber, and N. Kaempchen, "Lessons Learned from Automated Driving on Germany ' s Highways," *IEEE Intelligent Transportation Systems Magazine*, vol. 42, pp. 42–57, 2015.

[6] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards Fully Autonomous Driving: Systems and Algorithms," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 163–168.

[7] B. Arnaldi, R. Cozot, and S. Donikian, "Simulating Automated Cars in a Virtual Urban Environment 3 A Simulation Platform," in *Virtual Environments '95*, 1995, pp. 171–184.

[8] M. C. Figueiredo, R. J. F. Rossetti, R. A. M. Braga, and L. P. Reis, "An Approach to Simulate Autonomous Vehicles in Urban Traffic Scenarios," in *IEEE Conference on Inteligent Transportation Systems*, 2009, pp. 322–327.

[9] T. Gu and J. M. Dolan, "A Lightweight Simulator for Autonomous Driving Motion Planning Development," in *Proceedings of the International Conference on Intelligent Systems and Applications*, 2015, pp. 94–97.

[10] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual Worlds as Proxy for Multi-Object Tracking Analysis," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4340–4349.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA : An Open Urban Driving Simulator," in *Conference on Robot Learning (CoRL)*, 2017, p. 16.

[12] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving Policy Transfer via Modularity and Abstraction," in *Conference on Robot Learning (CoRL)*, 2018, p. 15.

[13] "AI Simulator for Self-Driving Development," aimotive.com, accessed: 2019-02-04.

[14] "NVIDIA Drive Constellation - Virtual Reality Autonomous Vehicle Simulator," www.nvidia.com/en-us/self-driving-cars/drive-constellation, accessed: 2019-02-04.

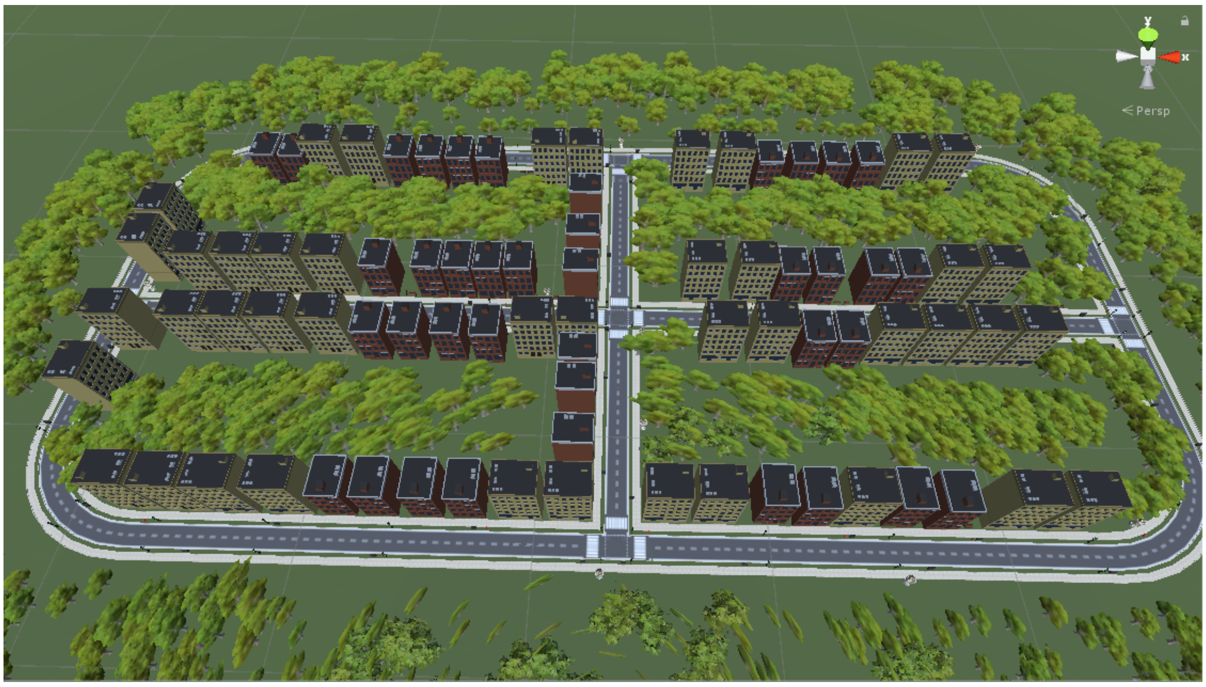
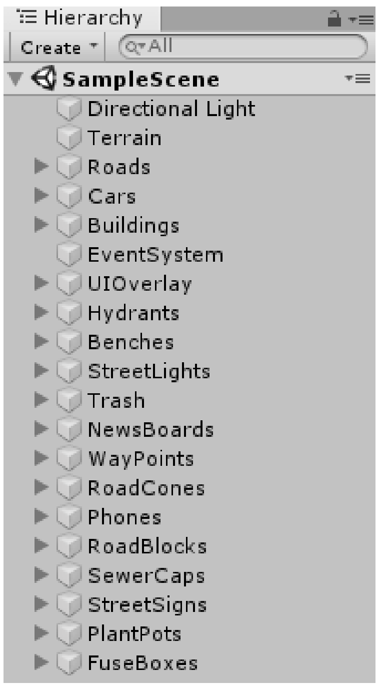


Fig. 4: View of the complete city



Hierarchy
Create
All
SampleScene
- Directional Light
- Terrain
- Roads
- Cars
- Buildings
- EventSystem
- UIOverlay
- Hydrants
- Benches
- StreetLights
- Trash
- NewsBoards
- WayPoints
- RoadCones
- Phones
- RoadBlocks
- SewerCaps
- StreetSigns
- PlantPots
- FuseBoxes

Fig. 5: The full Unity hierarchy tab

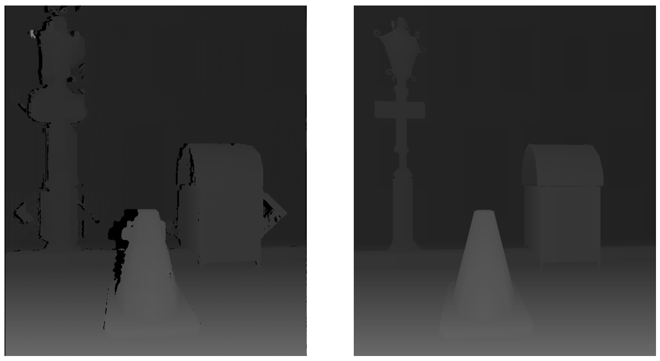


Fig. 6: Disparity and ground truth for the first example, SGM yielded accuracy of 86%

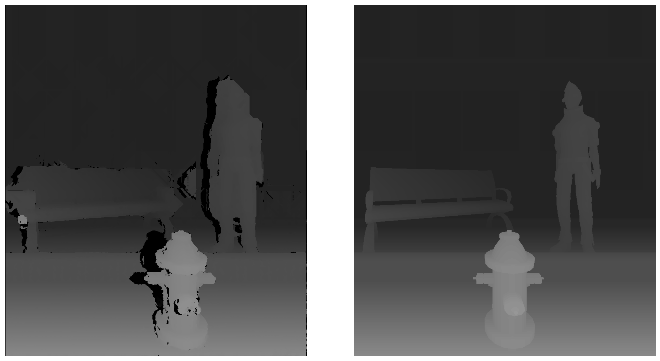


Fig. 7: Disparity and ground truth for the second example, SGM yielded accuracy of 85%

[15] "Apollo Simulation," apollo.auto/platform/simulation.html, accessed: 2019-02-04.

[16] "Cognata - Deep Learning Autonomous and ADAS Simulation," www.cognata.com, accessed: 2019-02-04.

[17] "CVEDIA SynCity," syncity.com, accessed: 2019-02-04.

[18] "RightHook RightWorld," righthook.io, accessed: 2019-02-04.

[19] "rFpro ADAS and Supervised Learning Autonomous Driving," www.rfpro.com, accessed: 2019-02-04.

[20] H. Hirschmueller, "Stereo Processing by Semiglobal Matching and Mutual Information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 328–341, 2008.

[21] "Unity User Manual," docs.unity3d.com/Manual/, accessed: 2019-02-04.

[22] S. Martull, M. Peris, and K. Fukui, "Realistic CG stereo image dataset with ground truth disparity maps," *Technical report of IEICE. PRMU*, vol. 111, no. 430, pp. 117–118, 2012.