UnrealCV: Connecting Computer Vision to Unreal Engine

Weichao Qiu(✉) and Alan Yuille

Johns Hopkins University, Baltimore, MD, USA
qiuwch@gmail.com, alan.l.yuille@gmail.com

Abstract. Computer graphics can not only generate synthetic images and ground truth but it also offers the possibility of constructing *virtual worlds* in which: (i) an agent can perceive, navigate, and take actions guided by AI algorithms, (ii) properties of the worlds can be modified (e.g., material and reflectance), (iii) physical simulations can be performed, and (iv) algorithms can be learnt and evaluated. But creating realistic virtual worlds is not easy. The game industry, however, has spent a lot of effort creating 3D worlds, which a player can interact with. So researchers can build on these resources to create virtual worlds, provided we can access and modify the internal data structures of the games. To enable this we created an open-source plugin *UnrealCV* (Project website: http://unrealcv.github.io) for a popular game engine Unreal Engine 4 (UE4). We show two applications: (i) a proof of concept image dataset, and (ii) linking Caffe with the virtual world to test deep network algorithms.

1 Introduction

Computer vision has benefited enormously from large datasets [7,8]. They enable the training and testing of complex models such as deep networks [13]. But performing annotation is costly and time consuming so it is attractive to make synthetic datasets which contain large amounts of images and detailed annotation. These datasets are created by modifying open-source movies [2] or by constructing a 3D world [9,17]. Researchers have shown that training on synthetic images is helpful for real world tasks [11,14,16,18,21]. Robotics researchers have gone further by constructing 3D worlds for robotics simulation, but they emphasize physical accuracy rather than visual realism. This motivates the design of realistic *virtual worlds* for computer vision where an agent can take actions guided by AI algorithms, properties of the worlds can be modified, physical simulations can be performed, and algorithms can be trained and tested. Virtual worlds have been used for autonomous driving [5], naive physics simulations [1] and evaluating surveillance system [19]. But creating realistic virtual worlds is time consuming.

The video game industry has developed many tools for constructing 3D worlds, such as libraries of 3D object models. These 3D worlds are already realistic and the popularity of games and Virtual Reality (VR) drives towards even

G. Hua and H. Jégou (Eds.): ECCV 2016 Workshops, Part III, LNCS 9915, pp. 909–916, 2016.
DOI: 10.1007/978-3-319-49409-8_75

greater realism. So modifying games and movies is an attractive way to make virtual worlds [5]. But modifying individual games is time-consuming and almost impossible for proprietary games. Hence our strategy is to modify a game engine, so that all the games built on top of it can be used. We develop a tool, UnrealCV, which can be used in combination with a leading game engine, Unreal Engine 4 (UE4), to use the rich resources in the game industry. UnrealCV can also be applied to 3D worlds created for virtual reality, architecture visualization, and computer graphics movies, provided they have been created using UE4. More precisely, UnrealCV provides an UE4 plugin. If a game, or any 3D world, is compiled with this plugin then we can create a virtual world where we can access and modify the internal data structures. This allows us to connect AI programs, like Caffe, to it and use a set of commands provided by UnrealCV to obtain groundtruth, control an agent, and so on. Figure 1 shows a synthetic image and its ground truth generated using UnrealCV.

We stress that we provide an open-source tool to help create new virtual worlds, which differs from work which produces a single virtual world [5] or creates synthetic datasets [9,17]. We hope that our work can help build a bridge between Unreal Engine and computer vision researchers.

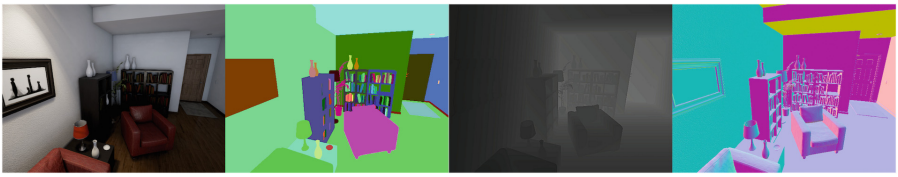

Fig. 1. A synthetic image and its ground truth generated using UnrealCV. The virtual room is from technical demo RealisticRendering, built by Epic Games. From left to right are the synthetic image, object instance mask, depth, surface normal

2 Related Work

Virtual worlds have been widely used in robotics research and many robotics simulators have been built [12,20]. But these focus more on physical accuracy than visual realism, which makes them less suitable for computer vision researchers. Unreal Engine 2 (UE2) was used for robotics simulation in USARSim [3], but UE2 is no longer available and USARSim is no longer actively maintained.

Computer vision researchers have created large 3D repositories and virtual scenes [4,6,10,15]. Note that these 3D resources can be used in the combined Unreal Engine and UnrealCV system.

Games and movies have already been used in computer vision research. An optical flow dataset was generated from the open source movie Sintel [2]. TORCS, an open source racing game, was converted into a virtual world and used to train an autonomous driving system [5]. City scenes were built [9,17] using the Unity

game engine to produce synthetic images. By contrast, UnrealCV extends the functions of Unreal Engine and provides a tool for creating virtual worlds instead of generating a synthetic image/video dataset or producing a single virtual world.

3 Unreal Engine

A game engine contains the components shared by many video games, such as rendering code and design tools. Games built using a game engine combine components from the engine with the game logic and 3D models. So modifying a game engine can affect all games built on top of it.

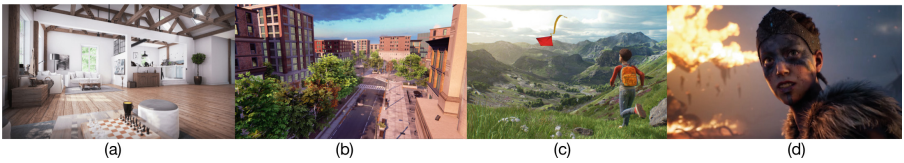

Fig. 2. Images produced by UE4, (a) (b) An architectural visualization and an urban city scene from Unreal Engine marketplace. (c) An open-source outdoor scene KiteRunner. (d) A digital human from the game Hellblade, shown in the conference GDC2016

We chose UE4 as our platform for these reasons: (I) It is fully open-source and can be easily modified for research. (II) It has the ability to produce realistic images, see Fig. 2. (III) It provides nice tools and documentation for creating a virtual world. These tools integrate well with other commercial software and well maintained. (IV) It has a broad impact beyond the game industry and is a popular choice for VR and architectural visualization, so high-quality 3D contents are easily accessible.

4 UnrealCV

UE4 was designed to create video games. To use it to create virtual worlds, a few modifications are required: (I) The camera should be programmably controlled, instead of by the keyboard and mouse, so that an agent can explore the world. (II) The internal data structure of the game needs to be accessed in order to generate ground truth. (III) We should be able to modify the world properties, such as lighting and material.

UnrealCV extends the function of UE4 to help create virtual worlds. More specifically, UnrealCV achieves this goal by a plugin for UE4. Compiling a game with the plugin installed embeds computer vision related functions to produce a virtual world. Any external program can communicate with this virtual world and use a set of commands provided by UnrealCV to perform various tasks. For example, the command `vget /camera/0/rotation` can retrieve the rotation of the first camera in the scene.

Architecture. UnrealCV consists of two parts. The first is the UnrealCV server, which is embedded into a virtual world to access its internal data structure. The second is the UnrealCV client whose function is provided by a library which can be integrated into any external program, like Caffe, enabling the program to send commands defined by UnrealCV to the server to perform various tasks. The architecture is shown in Fig. 3.

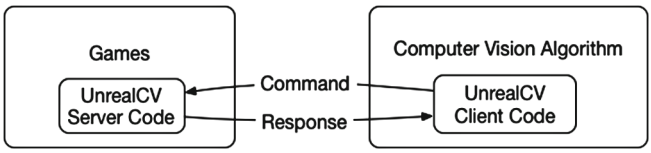

Fig. 3. The UnrealCV server is an UE4 plugin embedded into a game during compilation. An external program uses the UnrealCV client to communicate with the game.

The UnrealCV server is an UE4 plugin. After installing the plugin to UE4, the UnrealCV server code will be embedded into a game during compilation. The server will start when the game launches and wait for commands. The UnrealCV client uses a socket to communicate with the server. We implemented the client code for Python and MATLAB. Socket is a method of communicating between programs and is universal across programming languages and operating systems. So it is easy to implement a client for any language and platform that can support socket.

The server and client communicate using a plain text protocol. The client sends an UnrealCV command to the server and waits for a response. The command can be used to do various tasks. It can apply force to an object; can modify the world by changing the lighting or object position; can get images and annotation from the world. For example, the commands `vget /camera/0/image` and `vget /camera/0/depth` can get the image and depth ground truth. The command will save image as PNG file and return its filename. Depth will be saved as high dynamical range (HDR) image file, since the pixel value of PNG is limited within $[0 \ldots 255]$. The command `vset /camera/0/position 0 0 0` sets the camera position to [0, 0, 0]. An UnrealCV command contains two parts. The first part is an action which can be either `vget` or `vset`. The `vget` means getting information from the scene without changing anything and `vset` means changing some property of the world. The second part is an URI (Uniform Resource Identifier) representing something that UnrealCV can control. The URI is designed in a hierarchical modular structure which can be easily extended.

Features. The design of UnrealCV gives it three features:

Extensiblity: The commands are defined in a hierarchical modular way. Setting the light intensity can be achieved by `vset /light/[name]/intensity` to change the light color, a new command `vset /light/[name]/color` can be

added without affecting the existing commands. UnrealCV is open-source and can be extended by us or other researchers.

Ease of Use: Since we provide compiled binaries of some virtual worlds, such as a realistic indoor room, using UnrealCV is as simple as downloading a game and running it. Hence researchers can use UnrealCV without knowledge of UE4. The design supports cross-platform and multi-languages (Python, MATLAB). It is straightforward to integrate UnrealCV with external programs and we show an example with Caffe in Sect. 5.

Rich Resources: UnrealCV only uses the standard Application Programming Interface (API) of UE4, making it compatible with games built with UE4. We will provide virtual worlds with UnrealCV integrated and also host a model zoo to share virtual worlds created by the community.

5 Applications

In this section we created a virtual world based on the UE4 technical demo RealisticRendering[1] which contains an indoor room with sofa, TV, table, bookshelves, floor lamp, etc. The virtual world can be downloaded from our project website. We demonstrate two applications of this virtual world in this section.

Generating a Synthetic Image Dataset. We use a script to generate a synthetic image dataset from the virtual world. Images are taken using random camera positions. The camera is set to two different heights, human eye level and

Fig. 4. Images with different camera height and different sofa color. (Color figure online)

Algorithm 1. Generate a synthetic image dataset from a virtual world

```
vget /objects ;                                          // Get objects information
for all camera position do
    /* Set the virtual camera position                                          */
    vset /camera/0/location [x] [y] [z];
    vset /camera/0/rotation [yaw] [pitch] [roll];
    /* Get image and ground truth                                               */
    vget /camera/0/image;
    vget /camera/0/depth, vget /camera/0/object_mask;
end
```

[1] https://docs.unrealengine.com/latest/INT/Resources/Showcases/RealisticRendering/.

a Roomba robot level. The lighting, material property and object location can also be changed to increase the variety of the data, or to diagnose the strengths and weaknesses of an algorithm. Images with different camera height and sofa color can be seen in Fig. 4. Ground truth, such as depth, surface normals and object instance masks, is generated together with the images, shown in Fig. 1. The ability to generate rich ground truth is particularly useful for training and testing algorithms which perform multiple tasks and for detailed understanding of a scene. The UnrealCV commands used to generate this synthetic image dataset are shown in Algorithm 1. The synthetic images are on our website and a tutorial shows how to generate them step-by-step.

Diagnosing a Deep Network Algorithm. We take a Faster-RCNN model[2] trained on PASCAL and test it in the virtual world by varying rendering configurations. The testing code uses the UnrealCV client to control the camera in the virtual world and the Faster-RCNN code tries to detect the sofa from different views. We moved the position of the camera but constrained it to always point towards the sofa shown in Fig. 4. We got the object instance mask of the sofa and converted it into ground truth bounding box for evaluation. Human subject can easily detect the sofa from all the viewpoints. The Average Precision (AP) result shows surprisingly large variation as a function of viewpoint, see Table. 1. For each az/el combination, the distance from the camera to the sofa was varied from 200 cm to 290 cm. The symbol "-" means the sofa is not visible from this viewpoint. More generally, we can vary parameters such as lighting, occlusion level, and camera viewpoint to thoroughly test an algorithm.

Table 1. The Average Precision (AP) when viewing the sofa from different viewpoints. Observe the AP varies from 0.1 to 1.0 showing the sensitivity to viewpoint. This is perhaps because the biases in the training cause Faster-RCNN to favor specific viewpoints.

Elevation	Azimuth				
	90	135	180	225	270
0	-	0.713	0.769	0.930	0.319
30	0.900	1.000	0.588	1.000	0.710
60	0.255	0.100	0.148	0.296	0.649

6 Conclusion

This paper has presented a tool called UnrealCV which can be plugged into the game engine UE4 to help construct realistic virtual worlds from the resources of the game, virtual reality, and architecture visualization industries. These virtual worlds allow us to access and modify the internal data structures enabling us

[2] We use the implementation: https://github.com/rbgirshick/py-faster-rcnn.

to extract groundtruth, control an agent, and train and test algorithms. Using virtual worlds for computer vision still has challenges, e.g., the variability of 3D content is limited, internal structure of 3D mesh is missing, realistic physics simulation is hard, and transfer from synthetic images remains an issue. But more realistic 3D contents will be available soon due to the advance of technology and the rising field of VR. As an industry leader, UE4 will benefit from this trend. UnrealCV is an open-source tool and we hope other researchers will use it and contribute to it.

Acknowledgment. We would like to thank Yi Zhang, Austin Reiter, Vittal Premachandran, Lingxi Xie and Siyuan Qiao for discussion and feedback. This project is supported by the Intelligence Advanced Research Projects Activity (IARPA) with contract D16PC00007.

References

1. Battaglia, P.W., Hamrick, J.B., Tenenbaum, J.B.: Simulation as an engine of physical scene understanding. Proc. Nat. Acad. Sci. **110**(45), 18327–18332 (2013)
2. Butler, D.J., Wulff, J., Stanley, G.B., Black, M.J.: A naturalistic open source movie for optical flow evaluation. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012. LNCS, vol. 7577, pp. 611–625. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33783-3_44
3. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a robot simulator for research and education. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 1400–1405. IEEE (2007)
4. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al.: ShapeNet: an information-rich 3D model repository. arXiv preprint arXiv:1512.03012 (2015)
5. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: DeepDriving: learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2722–2730 (2015)
6. Choi, S., Zhou, Q.Y., Miller, S., Koltun, V.: A large dataset of object scans. arXiv preprint arXiv:1602.02481 (2016)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, pp. 248–255. IEEE (2009)
8. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The Pascal Visual Object Classes (VOC) challenge. Int. J. Comput. Vis. **88**(2), 303–338 (2010)
9. Gaidon, A., Wang, Q., Cabon, Y., Vig, E.: Virtual worlds as proxy for multi-object tracking analysis. arXiv preprint arXiv:1605.06457 (2016)
10. Handa, A., Patraucean, V., Badrinarayanan, V., Stent, S., Cipolla, R.: SceneNet: understanding real world indoor scenes with synthetic data. arXiv preprint arXiv:1511.07041 (2015)
11. Hattori, H., Naresh Boddeti, V., Kitani, K.M., Kanade, T.: Learning scene-specific pedestrian detectors without real data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3819–3827 (2015)
12. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2004), vol. 3, pp. 2149–2154. IEEE (2004)

13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
14. Marin, J., Vázquez, D., Gerónimo, D., López, A.M.: Learning appearance in virtual scenarios for pedestrian detection. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 137–144. IEEE (2010)
15. Mottaghi, R., Rastegari, M., Gupta, A., Farhadi, A.: "What happens if..." learning to predict the effect of forces in images. arXiv preprint arXiv:1603.05600 (2016)
16. Peng, X., Sun, B., Ali, K., Saenko, K.: Learning deep object detectors from 3D models. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1278–1286 (2015)
17. Ros, G., Sellart, L., Materzynska, J., Vazquez, D., Lopez, A.M.: The SYNTHIA dataset: a large collection of synthetic images for semantic segmentation of urban scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3234–3243 (2016)
18. Su, H., Qi, C.R., Li, Y., Guibas, L.J.: Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2686–2694 (2015)
19. Taylor, G.R., Chosak, A.J., Brewer, P.C.: OVVV: using virtual worlds to design and evaluate surveillance systems. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8. IEEE (2007)
20. Todorov, E., Erez, T., Tassa, Y.: MUJoCo: a physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033. IEEE (2012)
21. Vazquez, D., Lopez, A.M., Marin, J., Ponsa, D., Geronimo, D.: Virtual and real world adaptation for pedestrian detection. IEEE Trans. Pattern Anal. Mach. Intell. **36**(4), 797–809 (2014)