Large-scale Synthetic Data Generation of Ocean Marker Buoys with Unreal Engine 5

Anonymous CVPR submission

Paper ID *****

Abstract

Training of deep learning algorithms requires large amounts of data for them to perform well. This paper presents a method for generating large-scale sets of synthetic data using Unreal Engine 5. The method is discussed in detail, including development of the environment, objects of interest and data generation pipeline. The paper discusses the benefits of synthetic data generation in the context of maritime navigation as collecting real world data can be inconvenient or dangerous. This paper covers a two-stage pipeline using Unreal Engine 5.4's C++ classes for image generation and Python for labelling. Once set up, this method has the capacity to generate around 2000 pieces of labelled data (images and labels) per hour and provides source code for development of datasets for other purposes.

1. Introduction

Collecting real-world data for training object detection models can be a lengthy and inconvenient process, potentially incurring risk to researchers in some settings. Synthetic data can supplement real-world datasets or, in some cases, entirely replace them in model training.

The generation of synthetic data for machine learning tasks can save time, reduce risk and increase the scope of the data produced. One study identified a benefit in generating synthetic data of car crashes for predicting dangerous vehicles, reducing the missed detection rate by 18.5% compared to real data only models. [6]

However, synthetic data generation can often lead to data which is too clean looking and missing the imperfections of the real world, such as natural variation, deformation and clutter [3].

2. Synthetic Data Generation with Unreal Engine 5

To test the efficiency of this data generation method, a total of 1000 images were generated. With an initial delay of 5 seconds to allow the simulation to properly load and with a time taken of 2 seconds per image, the total time taken to generate the images was 33 minutes and 25 seconds. This does not include the time taken to generate the environment or the buoys, which was carried out separately. Furthermore, running the python script to generate the labels takes only a few seconds. This is covered in more detail in section 5.8.

At a resolution of 512x512, around 600MB in data was generated, consisting of labels and images with varying backgrounds, camera positions, lighting and weather conditions. This method frees up time for researches to focus on other tasks such as building the environments and sourcing realistic 3D models and textures.

Unreal Engine 5 allows for the creation of realistic environments in basically any setting, if one can source the correct assets. This allows for the generation of data which covers scenarios which might be difficult or impossible to capture in the real world. For example, it allows for the generation of data in dangerous conditions such as in a storm or natural disaster and inconvenient conditions such as specific times of day or year.

3. Methodology

3.1. Introduction

The synthetic data generation system was built using Unreal Engine 5.4 [5] and Python 3.11 [4]. The two systems do not run in unison and require running once each to generate the data, first the Unreal Engine simulation and then the Python script. The Unreal Engine simulation generates the images and the Python script generates the labels. While Unreal Engine sports a powerful blueprint system, C++ code was mainly used to develop the system such as event sequencing 5.2, camera positioning 5.3 and screenshot capture.

This was developed and tested on a machine running Windows 11 with an RTX 3080 GPU, i7 8700k CPU and 32GB of RAM with the storage being a Samsung SSD 970 EVO Plus 2TB.

3.2. Achieving High Fidelity

Aside from being an efficient method of generating synthetic data, the contents of the data can be completely controlled. Unlike content generated with some neural networks [8], there are no hallucinations to deal with and temporal consistency is maintained throughout the data.

3.3. The Terrain

Terrain, with the purpose of appearing in scenes (and not just as a requirement for fluid simulation as discussed in section 3.4), was created using one of two methods.

Two methods were experimented with for creating the terrain. The first was to make use of assets available on the Unreal Engine marketplace, most notably the Megascans library [7]. This method has the benefit of allowing the developer to create a realistic environment which can be travelled. However, a major downside is that assets often cost money and it can take hours, days, even weeks to create a realistic environment, depending on the type of environment needed.

Talk about heightmaps

The second method was to use high dynamic range images (HDRI) to create the illusion of terrain, which normally provided a realistic sky too as a bonus. This method allowed for near instant creation of a realistic environment, giving the illusion of entire lakes with mountains in the background. However, the downside is that as the image is projected in a dome around a central point, moving even small distances away from the central point can cause the illusion to be broken. For this task, HDRI-based environments were suitable as the camera was moving only short distances around a static point.

3.4. The Water

The water in the simulation is provided by Unreal Engine 5's Water plugin, more specifically the Water Body Ocean class. This water is simulated at interactive rates as a fluid, with the ability to simulate waves and the results of environment effects such as colour and lighting. With the fluid being simulated in real time, the water is able to be interacted with by entities (known as actors) in the scene, such as the buoys, as discussed in 4.1. Although HDRI environments were used, this plugin requires some terrain to be present in the scene to work.

Wave simulation is achieved through the use of a trochoidal or Gerstner wave function [1]. Parameters such as wavelength,wavelength falloff, amplitude, steepness, speed and direction can be set to create a range of wave types.

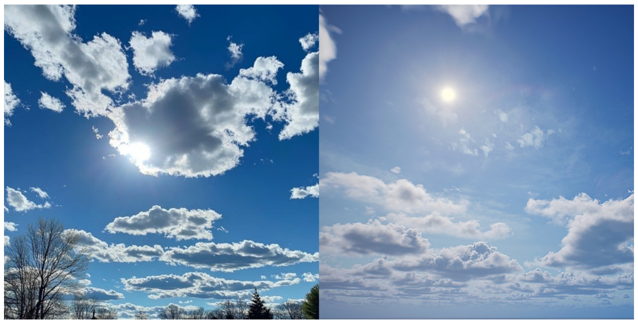

Figure 1. A real photo of a sky (left) and Ultra Dynamic Sky (right).

The water can also be set to be reflective, refractive and to have caustics, although it was noted that this had a negative impact on frame rate.

3.5. The Sky

As with the terrain, two methods were experimented with for generating a realistic sky. The first was to use a plugin called Ultra Dynamic Sky [2] which provided scene-wide realistic sky scapes and weather conditions. The second was to use a high dynamic range image (HDRI) to create the illusion of a sky. This method works well for static scenes in which minimal movement is required. Figure 1 shows a comparison between a real sky and the Ultra Dynamic Sky plugin. Images courtesy of stockcake.com [9] and the Ultra Dynamic Sky plugin respectively.

3.6. The Weather

Weather conditions, such as rain, snow and fog were also generated with the Ultra Dynamic Sky plugin, which includes a package called Ultra Dynamic Weather. As with the sky, this plugin allows for the generation of a range of weather scenarios which can be controlled. This was the only method used for generating weather conditions as it was the most efficient and realistic method found. Unlike with a HDRI, this method is not location specific and allows for weather simulation at any position within the scene.

4. The Buoys

4.1. The Buoys

The buoys are included in the scene as actors with static meshes, which play the part of digital twins of real marker buoys. The 3D models were purchased and imported into the scene. A class of MarkerBuoyBase was created which was inherited by all buoys in the scene. This class contained the logic for the buoys' interactive behaviour.

Each buoy was placed in the scene at the same location to allow for the HDRI background to be effective, with all buoys apart from the subject of the current visit being set to invisible.

4.2. Buoyancy

Unreal Engine 5's water plugin allows for buoyancy simulation through the use of virtual pontoons. They act in the scene like invisible floatation aides and can be attached to objects allowing for fine tuning of the buoyancy of an object. The pontoons can be given X, Y and Z coordinates, relative to the object they are attached to, along with a buoyancy force.

For each buoy, 4 pontoons were placed at relative coordinates of:

- X: 300, Y: 300, Z: 150
- X: 300, Y: -300, Z: 150
- X: -300, Y: -300, Z: 150
- X: -300, Y: 300, Z: 150

5. The Data Generation Pipeline

5.1. Introduction

Having a realistic environment is great and can be a lot of fun to make. However, it does not achieve the end goal of this project. Actually having data which can be used to train models, with minimal input from users is the goal. This section discusses the pipeline, dubbed SyntheticDataGenerationSystem, which was a C++ class included as an actor in the scene and used to generate the data.

SyntheticDataGenerationSystem worked by looping through all of the actors in the scene which share the parent class of MarkerBuoyBase. The system effectively works by setting all buoys to the colour red 5.6, hiding all of the buoys, positioning the camera randomly near the buoy and facing it, showing the buoy, taking a screenshot, changing its colour from red to its normal texture by changing its render pass in real time, taking a second screenshot and then hiding the buoy again. This process is repeated for each buoy in the scene. Once all of the images have been created, a Python script 5.8 is run which gets the coordinates of the outline of the buoy in the image and stores them in a .txt file.

5.2. Handling Concurrent Actions

With the simulation being run in a video game engine, everything happens at interactive rates based on the frame rate of the simulation. Initially, a fundamental misunderstanding of how game engines work hindered progress. To summarise, as soon as the simulation began, all of the code written was executed in a single frame, causing n number of screenshots to be taken in an instant. The correct solution was to make use of Unreal Engine's Timer Handle (FTimerHandle) function, to schedule the taking of screenshots at a set interval.

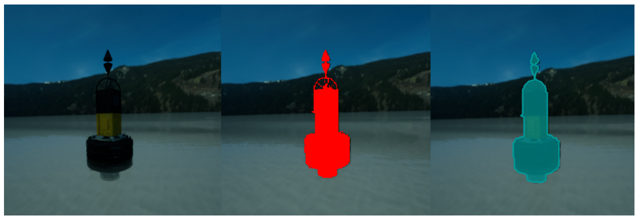
Figure 2. A buoy in a scene showing the stages of the mask generation.

To schedule the events, a cumulative delay was added to the timer handle in the form of:

cumulativeDelay = initialDelay + (markerBuoyNumber * numberOfVisits + visitNumber) * visitInterval;

5.3. Positioning the Camera

To replicate the position of a camera as if it were on some sort of vessel, randomisation of the camera's position was included. One limitation of this approach was that it was technically possible for all of the images to be taken from the same position, producing ineffective data for generalisation. However, this was not noticed in the data generated.

5.4. Semantic Labelling

Two methods for generating labels were tested. The first was to generate bounding boxes around the buoys in the scene using the inbuilt head's up display (HUD) feature. The second was to generate masks of the buoys in the scene.

5.5. Bounding Boxes

Bounding boxes were generated by detecting points on the screen, per frame, which intersected with the buoy's static mesh. The bounding box was then generated by drawing a rectangle around the most northern, easter, southern and western points on the screen (see figure 2). This method proved to be ineffective in generating tight bounding boxes around the buoys.

5.6. Semantic Segmentation Masks

To set the buoys as a uniform and unique colour, Unreal Engine 5's Custom Depth feature was used. One can liken this to the layer feature in software such as Adobe Photoshop. This feature allows a static mesh to be rendered in a separate render pass, with this render pass (or "layer") having a separate material applied to it. This feature was used to render the chosen buoy in just the colour red (RGB: 255, 0, 0).

Inside the material, a Scene Texture node was added with a Scene Texture ID of PostProcessInput0, followed by the creation of a second Scene Texture node with a Scene Texture ID of CustomStencil.

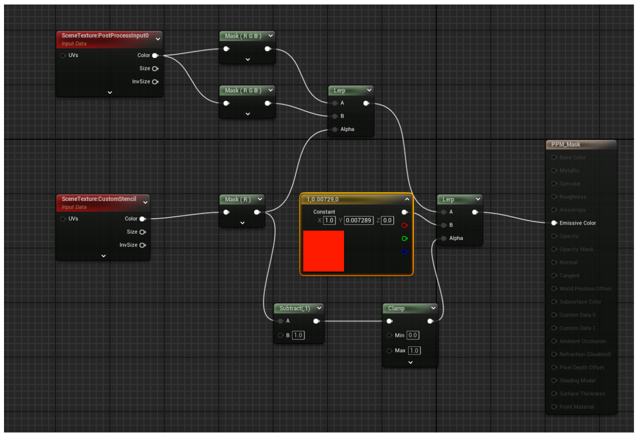

Figure 3. An image of the blueprint used to generate the material for all objects in the chosen render pass.

The RGB values of the colour (color) output of PostProcessInput0 were connected to a Linear Interpolation node's A and B input values, with the R value (just red) of CustomStencil's colour output connected to the Alpha input of the Linear Interpolation node.

The R value of CustomStencil was also connected to a Subtract node's A input, with the B input set to 1.0. The output of the Subtract node was connected to a Clamp node with the Min set to 0.0 and the Max set to 1.0. The output of the Clamp node was then connected to the Alpha input of a second Linear Interpolation node.

A 3 Vector Constant node with its RGB values set to 1.0, 0.0, 0.0 was connected to the B input of the second Linear Interpolation node. The output of the first Linear Interpolation node was connected to the Alpha input of the second Linear Interpolation node. Finally, the output of the second Linear Interpolation node was connected to the Emissive Colour input of the material. This material should then be applied to every static mesh with a CustomDepth Stencil Value of 2.

5.7. Scene Capture

By making use of FTimerHandle, as mentioned in 5.2, screenshots could be taken at set intervals. For each visit, two screenshots per buoy were taken, the first with the red semantic segmentation mask applied and the second without, with each being stored in separate directories. The position of the buoy was frozen for this so that the mask's coordinates perfectly matched that of the buoy in the unmasked image.

5.8. Storing Mask Coordinates with a Python Script

A Python script, (Mask Pixel Joiner), exports each image inside the masked screenshot directory, the script would open the image and scan each row from left to right, until a red

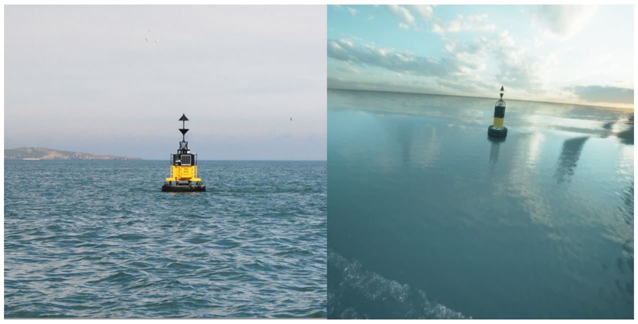

Figure 4. A real photo of a buoy (left) and a buoy generated in Unreal Engine 5 (right).

pixel was found. The script would then store the x and y coordinates of the red pixel. This would continue for every row of pixels. Once this was completed, the same process would be carried out but from bottom to top and right to left, determining the pixels which made up the edge of the buoy in an anti-clockwise direction. The coordinates were then stored in a .txt file.

6. Discussion

Applications Limitations Usecases

The main application of this work is for generating data to train object detection models.

7. Conclusion

In conclusion, this method for generating data is efficient and allows for scenarios to be catered for which might be difficult or impossible to capture in the real world. The data generated is of high fidelity and has high potential for use in training object detection models.

8. Future Work

Some sort of fail safe to ensure that the generation can recover if some issues occur such as a power cut.

Data augmentation

References

[1] Adrian Constantin and Stephen G Monismith. Gerstner waves in the presence of mean currents and rotation. *Journal of Fluid Mechanics*, 820:511–528, 2017. 2

[2] Everett Gunther. Ultra Dynamic Sky, 2025. Accessed: 07-Mar-2025. 2

[3] Chengjian Feng, Yujie Zhong, Zequn Jie, Weidi Xie, and Lin Ma. Instagen: Enhancing object detection by training on synthetic dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14121–14130, 2024. 1

[4] Python Software Foundation. Python 3.11.0 release, 2022. Accessed: 2025-03-10. 1

[5] Epic Games. Unreal engine 5.4 is now available, 2024. Accessed: 2025-03-10. 1

[6] Hoon Kim, Kangwook Lee, Gyeongjo Hwang, and Changho Suh. Crash to not crash: Learn to identify dangerous vehicles using a simulator. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):978–985, 2019. 1

[7] Quixel Megascans. Quixel Megascans: Photorealistic 3D Asset Library, 2025. Accessed: 07-Mar-2025. 2

[8] Stephan R. Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement, 2021. 2

[9] StockCake. Captivating free images for every occasion, 2025. Accessed: 2025-03-10. 2