DOI: https://doi.org/10.46354/i3m.2019.dhss.001

DEEP LEARNING OF VIRTUAL-BASED AERIAL IMAGES: INCREASING THE FIDELITY OF SERIOUS GAMES FOR LIVE TRAINING

Dean Reed[a], Troyle Thomas[b], Shane Reynolds[c], Jonathan Hurter[d] , Latika Eifert[e]

[a],[b],[c],[d] Institute for Simulation and Training, University of Central Florida
[e] U.S. Army Futures Command, Combat Capability Development Command-Soldier Center

[a] dreed@ist.ucf.edu, [b] tthomas@ist.ucf.edu, [c]sreynold@ist.ucf.edu, [d]jhurter@ist.ucf.edu
[e]latika.k.eifert.civ@mail.mil

ABSTRACT

The aim of rapidly reconstructing high-fidelity, Synthetic Natural Environments (SNEs) may benefit from a deep learning algorithm: this paper explores how deep learning on virtual, or synthetic, terrain assets of aerial imagery can support the process of quickly and effectively recreating lifelike SNEs for military training, including serious games. Namely, a deep learning algorithm was trained on small hills, or berms, from a SNE, derived from real-world geospatial data. In turn, the deep learning algorithm's level of classification was tested. Then, assets learned (i.e., classified) from the deep learning were transferred to a game engine for reconstruction. Ultimately, results suggest that deep learning will support automated population of high-fidelity SNEs. Additionally, we identify constraints and possible solutions when utilising the commercial game engine of Unity for dynamic terrain generation.

Keywords: synthetic natural environment, live military training, deep learning, simulation fidelity

1. INTRODUCTION

For positive training transfer to occur in a military simulation, the highest possible representation of the natural terrain must be achieved. To exemplify the train-as-we-fight mantra, any scenario represented by game engines needs a high, consistent correlation with the Real World (RW). The effective-and-efficient creation of terrain that meets accuracy requirements for live (such as augmented) military training environments continues to be a significant challenge. Thus, the purpose of this paper is to aid in the rapid reconstruction of Synthetic Natural Environments (SNEs) for training, by simultaneously maintaining both effectiveness (i.e., high fidelity) of the final environment and efficiency (i.e., a low amount of resources) within the reconstruction process. To reach this purpose, the first-tier goal of the paper is multifold: to test the classification accuracy of a deep learning algorithm trained on a SNE, and test the subsequent reconstruction of a SNE based off the classification from the deep learning algorithm. A second-tier goal is to identify constraints in a game engine's terrain generation process, as lessons learned for future research.

1.1. Related Work

Representing RW or geo-specific training regions to a very high degree of RW correlation, while maintaining cost-effectiveness, has been elusive. In one method, scenario environments can be painstakingly built by artists—reducing efficiency and introducing a potential human-error confound to effectiveness. Some alternate solutions have been proposed: one example is the automatic generation of terrain from aerial photogrammetry (Spicer, McAlinden, and Conover 2016). With the advent of low-cost and highly-reliable drone platforms, obtaining dense point-cloud data from either photogrammetric or LiDAR sources has become commonplace. However, processing the dense point cloud into usable and efficient polygonal, game-engine-based geospecific regions of sufficient geographic size to perform on-going mission training remains a challenge. One issue is the high-resolution automatic classification of different features (e.g., capturing the distinction between human-made structures and natural terrain). We distinguish this high-resolution classification from the traditional coarse-grain approach of Digital Terrain Model (DTM) and Digital Surface Model (DSM) methods commonly provided by Commercial-Off-The-Shelf (COTS) software tools. Furthermore, injection of numerous raw or coarse point-cloud-derived polygonal models into the game-engine-based rendering system is both inefficient and distracting for the trainee.

The U.S. Army's One World Terrain (OWT) program is a large program attempting to solve these terrain-related issues. The goal of OWT is to provide foundational, attributed 3D data to runtime publishers (i.e. consumers) that is well-formed and consumable at the Point-Of-Need (PON). OWT will contain polygonal mesh OBJ data and a traditional-gridded CDB standard (e.g. elevation grids, imagery, and vectors). OWT data should be viewable and editable with no proprietary tools required. OWT will leverage Open Geospatial Consortium (OGC) CDB to add rigour, structure, attribution and determinism to OBJ files. Nevertheless, machine learning (via a deep neural network) to classify and reconstruct objects from aerial imagery point clouds has been introduced (Zhang, Li, Li & Liu, 2018). Although the latter avenue of work supplies a classification process, it is limited by training

a deep learning algorithm through non-virtual data, which is a problematic method from the vantage of efficiency.

In the advancement of machine learning, previous research suggests the value (in terms of efficiency and effectiveness) of using virtual, or synthetic, imagery datasets to train object classification of RW imagery: a deep learning algorithm was successfully tested for classification after training on a virtual military helmet (Reed, Thomas, Eifert, Reynolds, Hurter, and Tucker 2018). Although virtual assets have been used to help pre-train for aerial images (Kemker, Salvaggio, and Kanan 2018), or enhance the training process (Chen, Jiang, Li, Jia, and Ghamisi 2016), it is unclear how virtual assets would fair as an exclusive training set for aerial images. Ultimately, the present paper investigates how the artificial-intelligence technique of deep learning can be leveraged to alleviate the burden of having to populate repetitive, but critical, terrain features in a SNE. We also identify and explore shortcomings in the Unity game engine as used to render SNEs.

1.2. Solution Space

The proposed solution for SNE reconstruction is to leverage deep learning to help classify features of an environment; in turn, the classified features can then be utilised to populate a SNE with virtual models and textures rapidly. The focus of this paper's initial test is the classification of berms or small hills (see Figure 1) found in a SNE. For streamlining, the solution is directed at training the deep learning algorithm using virtual, rather than RW, imagery data.

Figure 1: Example of a Berm in a Synthetic Natural Environment (SNE; Image Contrast Increased for Presentation)

One can train a deep learning algorithm to classify geo-specific features from aerial imagery. Once features are learned, one may inject matched features with a high degree of accuracy into the SNE. Given how large defence organisations generally have a preexisting set of high-quality terrain models, which are used to populate geotypical and geospecific SNEs, deep learning serves as a bridge to support automatic modelisation of simulated military environments. Specifically, the U.S. Army has a large resource of assets: with a large set of Synthetic Environment (SE)-Core 3D models library called the common models, the Army can access Games for Training assets (e.g., the Virtual Battlespace game engine or the Army Model Repository; PEO STRI n.d.; Baker 2018). The availability of thousands of 3D models is beneficial for training deep learning algorithms, since these 3D models are existing assets usable for AI training material. This process differs from performing very expensive data collection of RW imagery.

Another linchpin of the solution is the capabilities of game engines. Commercial game engines, such as Unity, offer advanced features for rendering high-quality, natural-looking environments. Thus, game engines provide an apt tool to maintain ecological validity for various military tasks, due to the possibility of high fidelity afforded. However, a restriction arises with game engines: although some engines support the import of dense point-cloud data directly for small regions, the same process does not lend itself to efficient rendering over larger areas. Additionally, native dense point-clouds will inherently contain surface inaccuracies from the source sensor that causes the data to be unusable for traditional simulation. These game engine limitations underscore the need for a more efficient process for large-scale SNE development.

The foreseen reconstruction solution would ideally be routine for an Unmanned Aerial System (UAS), or drone. Once a deep learning system is trained, a UAS could collect data in the form of photos or LiDAR data, represented by a 3D point cloud. Based on the models derived from the point cloud, features would be identified, classified, and then mapped to a terrain object. Once an area is appropriately classified by the deep learning algorithm, it can be seamlessly imported into a game engine, such as Unity.

2. METHODOLOGY

Before detailing the experimental procedure, the second-tier goal of this paper will be covered: the constraints confronted (and solutions found) when attempting to use Unity for dynamic terrain generation.

2.1. Challenges in Unity

Virtual environment developers have a decision to make when deciding on how to implement terrain: in Unity, one option would be to use the built-in terrain editor, whereas another option would be to import a polygonal mesh from another tool. In Unity's 2018.3.8.f1 release (Tchou 2018), updated terrain implementation supports seamless operations between multiple co-located terrain tiles. The native Unity terrain implementation is beneficial, as it allows designers to quickly add content to terrain by applying terrain brushes that paint geotypical content on top of terrain tiles. Another benefit of Unity's native terrain is the relative efficiency of how repeated objects are rendered at runtime. Native implementations of objects, such as trees that are directly handled en masse (in contrast to individually-placed game objects) leads to superior handling in the draw cycle and better overall performance.

An internal preliminary experiment tested the implementation of native Unity terrain trees (i.e., terrain objects) versus individual trees placed as game objects. An individual, traditional 3D tree model was created with three levels of detail. Using the Unity terrain tool, 13,000 trees were painted over a single terrain tile. We then repeated the test but used game objects instead of a traditional 3D model, and placed them at the exact location of each tree in the first test. Locations of the trees were determined by the treeInstances method of the TerrainData Unity object. The Frames-Per-Second (FPS), or framerate, measurements were provided by the FRAPS program. A PC with a dual video card was used to perform this test, with consistent specifications (see Table 1).

Table 1: Testbed Specifications

CPU	Intel Core i7-5960X
Memory	32GB
OS	Windows 10 Enterprise
GPU	2x NVidia GeForce GTX 980 Ti

Ultimately, the native terrain condition used less memory and rendered more efficiently than the game object approach (see Table 2).

Table 2: Terrain Objects Vs. Game Objects, Tree Models

Object Type	Memory	FPS	CPU
Native Terrain	15%	36	6.6%
Game Object	16%	33	12.7%

As the results show, using the Unity terrain system to manage the tree models is more efficient, with respect to computer resources, and provides superior rendering performance.

Despite the Unity game engine's beneficial aspects within the terrain editor, various challenges inherent in using the native Unity terrain processes were found. These issues altered the final methodology used, and are discussed here as part of the methodological rationale. Ultimately, these choices may inform future research when considering the value of Unity's impact on serious military games.

Originally, a detail-mesh placement-map system was tested, where pixel data (in the form of RGB values) corresponded to different mesh objects. When placing terrain objects, manually setting the rotation of objects was unsupported. The system lacked the option for correcting the direction objects faced, on a per-object basis. However, the objects could be instructed to face the camera consistently (i.e., an option for billboarding existed). Further, the scaling of mesh objects was limited; this is a fundamental difference between injecting native terrain features and injecting representative game objects into an environment: typically, a game object can be used repeatedly with various scales. In contrast, Unity's native terrain-feature scales are set uniformly, breeding inefficiency: to account for scale variations of terrain, different-size versions of the same object would be needed, in turn requiring multiple maps. Another crucial control issue with the aforementioned map process was the non-deterministic placement of models: when an identical map was reloaded, the locations of models were not necessarily identical to the previous load. A similar non-deterministic issue involved the terrain brush settings for painting pixels on the map, due to the inability to consistently equate one model to one pixel. If the brush size was too large, multiple objects were placed; but if the size was too small, there was a chance no objects would be placed. Further errors may also be caused here if a user changed the settings on detail resolution.

Another issue within the map used was the lack of collision between objects. The lack of collisions is problematic since it reduces the fidelity of the environment: although the visuals may look real, they would not function realistically. For example, an avatar could walk through a berm or shoot a bullet through a berm; this defeats the purpose of the current project's goal (at least for some types of ground-based training). The detail-map size must also be consistent with the terrain size: if the sizes did not match, the terrain system would attempt to rescale the map, which skewed the placement of an object. Finally, there was a limit of objects allowed to be placed into a patch (i.e., an area of terrain) in Unity. It appears an upper-limit was based on vertices; given our terrain settings, this was an issue.

A limiter of fidelity was also calculated for the Unity environments. The most common way to attribute height vertices is to import a grayscale heightmap that represents individual heights of the terrain as a two-dimensional array of 16-bit grayscale values. This means that a single terrain tile can have a maximum representative range of 2^{16} or 65,536 discrete values at any one location in the 2D array. Current implementations of Unity support a maximum density of 4,096+1 for the heightmap. This limitation means that any grayscale-based terrain region represented by Unity's system must fit within this restriction. Selecting our terrain's ideal resolution to be of 1cm quality for each of our possible grayscale values, we find that our lowest-to-highest range spans a maximum distance of 655.36m for a single terrain tile. If we are representing an urban, fairly flat environment, this limitation is less concerning. Representing hilly, or excessively rugged terrain would require subdividing the tiles into very small segments. We summarise the relationship between desired game engine representation, in terms of representation fidelity of the terrain, as:

$$Ed = \frac{F}{2^{16}} \quad (1)$$

$$Ma = (4{,}097F)^2 \quad (2)$$

In equation (1) Ed is the elevation delta with the fidelity of F. Equation (2) shows the relationship between the maximum area, Ma, and the desired fidelity of representing the area expressed as a function of the

maximum Unity grayscale heightmap and the desired representational fidelity. Overall, the limitation of the 16-bit representation of depth and 4,097*4,097 resolution limitation can be constraining when attempting to preserve the highest possible RW representation within the game engine. The University of Central Florida Institute for Simulation and Training (or simply, IST) has begun implementation of a Unity terrain importer capable of preserving the original sensor fidelity. The importer bridges the gap between traditional geospatial source data and Unity terrain objects. The importer ingests 3D formats generally associated with 3D models (including OBJ, FBX, and DAE), which are then translated into the native Unity terrain tiles.

2.2. Experimental Procedure Summary

The overall procedure for the experiment followed a linear, stepwise process:

1. The physical data of the RW environment were collected via images.
2. The physical data of the RW environment were used to generate a point cloud.
3. The point cloud was converted to polygonal meshes.
4. The polygonal meshes were imported into the Unity game engine as part of a SNE.
5. The SNE, as a section, was used as training fodder for a deep learning model.
6. The deep learning model's decisions were used to infer and place game objects into the remaining section of the SNE.

2.3. Data Collection Process

Sensors, such as LiDAR and software photogrammetric techniques, are established sources for dense point-cloud collection. The dense point-cloud data sources can be fused from low-flying aerial platforms, ground systems, or satellite-based multi-spectral sensors. For the present dataset, a Man-Wearable System (MWS) and a low-flying drone were leveraged to generate a high-density point-cloud representation of a real Army training range, located at Aberdeen Proving Grounds, Maryland.

2.3.1. Drone Data Capture

The U.S. Army Futures Command, Combat Capability Development Command-Soldier Center (known as the Simulation and Training Technology Center (STTC)) and IST, in cooperation with a small business, Micro Aerial Projects, implemented a medium-frame UAS for high-quality data acquisition. IST assembled, tested, and instrumented a quad-propeller, semi-autonomous UAS for rapid data collection that complied with Federal Aviation Administration Guidelines (Part 107) and implemented full control over both part source and traceability of the onboard PixHawk flight controller. The overriding goal was to collect the highest quality, georeferenced photos as possible. IST leveraged the Micro Aerial V-Map system along with the highest quality camera available at the time, the SONY-A7RII. The V-Map system, which leveraged Real-Time Kinematic (RTK) GPS, was used to collect the large majority of the RW training range. The V-Map system allows correlation accuracy of 10mm on the horizontal axis and 15mm on the vertical axis (Micro Aerial Projects L.L.C. n. d.).

IST flew the UAS at the height of 50 meters, then again at the height of 40 meters, to collect the imagery in the focal length of the sensor and lens configuration of the A7R-II. Each run was flown in orthogonal vectors to ensure maximum sensor overlap.

2.3.2. Man Wearable System (MWS)

To supplement the drone capture, IST built a man-wearable photogrammetry system: using the MWS (see Figure 2), areas that were inaccessible by traditional drone collection method were able to be collected. The MWS enabled data collection under low-hanging canopies, under power lines, and around interiors. The system auto-triggers the Sony A7R-II camera based on movement derived from a Pixie RTK GPS. The percentage of overlap between photos can be entered by the user into the mobile-computing platform's display. Based on the requested overlap, the georeferenced photos are automatically triggered by the MWS software, based on the distance trajectory being tracked on the embedded computer. The MWS was based on a gimbal and had a jitter-eliminating pendulum and metal-armed frame to reduce motion blur introduced by normal walking. The images obtained with the MWS are georeferenced by recoding the RTK GPS location, heading, velocity, and time attributes along with the photographs. Accuracy was improved by including ground control points to minimise registration and camera trajectory errors in the post-processing software. Both PhotoScan and RealityCapture software were used to generate photogrammetric-derived dense point-clouds for merging into the dataset provided by the UAS.

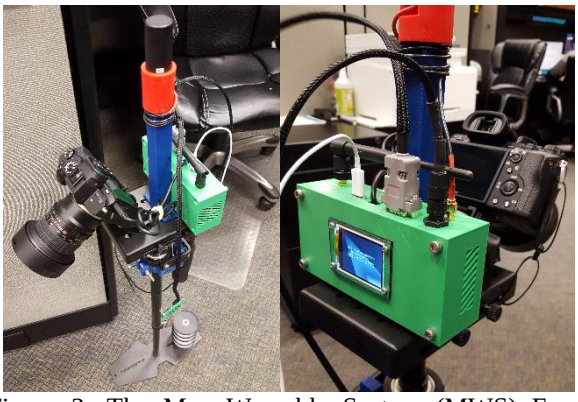

Figure 2: The Man Wearable System (MWS) From Different Angles

2.4. Dense Point Cloud to Polygonal Mesh Conversion

To visualise point-cloud data in a game engine, the generated point cloud must be processed into a polygonal mesh. Unfortunately, the resultant initial mesh can be extremely dense, leading to very realistic but less

ISBN 978-88-85741-33-1; Bruzzone and Sottilare Eds.

efficient—terrain (especially on mobile or embedded training devices). For example, one berm at full resolution could be represented by over 25,000 triangles, once converted from a dense point-cloud into a mesh. Having multiple berms across a terrain would quickly hinder the framerate.

Typically, the quality of the final model is increased when beginning with a very dense model and working down to a low polygon model. As part of the optimisation process, the full resolution mesh was decimated until an acceptable balance between polygon count and fidelity was reached. Textures and normal maps created from the high-resolution mesh can aid in keeping a visually realistic model with a low polygon count. Ultimately, decimating in a stepwise fashion (e.g. decimating by 10% five times, instead of 50% one time) was found to produce a higher quality and more accurate decimated mesh. There are times that the high-resolution mesh needs to be re-topologised to allow for a quality decimation process; this is especially true if you have parts of a mesh that need to move, such as a door or a person's facial features. Sometimes the UVs (or two-dimensional texture coordinates) require manual modification in an external program, such as Photoshop.

2.5. Terrain Vs. Game Object Vs. Game Object Modelisation

Based on the desired use case, there are technical limitations to consider when trying to develop the high-fidelity environments needed for live training. Even though our data collection process allowed us to create an extremely detailed mesh that represents the real world very well, using all of that data in a training system is not realistic, because of computer performance implications. Therefore, a design choice includes replacing mesh objects (e.g., trees, buildings, vehicles), which were generated from point-clouds of the RW, with similar highly-optimised models. Ideally, a system could learn and recognise objects using RW data, find the best object replacement from a catalogue of optimised models, and place-and-fit the model correctly. These optimised models would fall somewhere between geotypical and geospecific, and would thus be geospecifical. This novel system would learn-and-classify streamed RW data, and then populate a training environment with highly optimised geospecifical models that closely match the RW data. This system would allow for the rapid creation of high-fidelity environments that are optimised to run on training devices.

2.6. Essential Model Training

A deep learning model was built to detect berms in the SNE automatically. For deep learning, the berms in the SNE were tagged and localised in order to build a dataset of ground-truth berms. Within Unity, berms were tagged with the berm class name and a game-object bounding box: an automated training session captured images of these objects at several different positions. Since the end application necessitated aerial detection of these objects, our training session captured dataset images from an aerial view. Ground-truth labels were also generated by calculating the image coordinates from the game objects' world coordinates.

After the SNE dataset was generated, the dataset was segmented into training, validation, and testing subsets: the percentage splits were 60:20:20 over a dataset of 267,300 images. The segmentation was used to provide datasets from the same domain that could be used for testing, validation, and training of the model. This segmentation prevents the model from being tested on images from which the model was originally trained. If a model is both trained and tested on the same dataset, then the model would be influenced to memorise the dataset.

A secondary dataset, which removed all game-object models from the scene, was also generated in order to identify any influence stemming from the additionally placed models.

The next step was to train the deep neural network model. In this case, the You Only Look Once (YOLOV3) model proved best, due to both its end-to-end network and its speed during inference. The model was trained using standard hyper-parameters provided in the YOLOV3-608 network configuration. Afterwards, the learned weights were visually tested for accuracy (see Figures 3 and 4 for an example of the detection). The findings suggested weights trained after 20,000 iterations provided reasonable results.

Figure 3: Example of Trained Model Output Detection

The selected weights were then evaluated on the testing subset of the generated dataset. The measure of performance of our model was determined through a precision-versus-recall curve. Precision indicates the ratio of correct detections as compared to total detections. Recall indicates the ratio of correct detections as compared to the total possible correct labels. Both precision and recall are functions of the intersection-over-union and a prediction-confidence threshold calculation. The intersection-over-union is the percentage of our two bounding boxes' (i.e., the ground-truth bounding box and the predicted bounding box)

overlap, over the total shared area. The resulting model's behaviour can be determined by plotting the results of precision and recall, as the intersection-over-union threshold is changed. The change in the threshold illustrates the design decision of requiring high precision, versus requiring more inclusivity, when using our trained model.

Figure 4: Example of Trained Model Output Detection

3. RESULTS

3.1. Precision and Recall Results

Figure 5 describes the behaviour of the trained model in terms of precision and recall for several trained weights when evaluated on a test dataset. The trained weights are labelled by the number of epochs of training that had occurred.

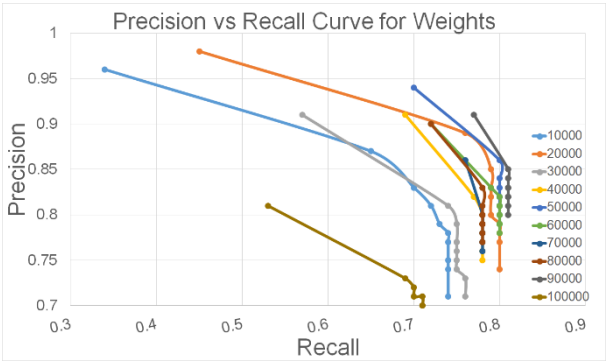

Figure 5: Graph of Precision and Recall, Showing Success of Deep Learning Algorithm

The 20,000 weight curve shows a model behaviour with a precision maximum of 0.98, but a corresponding recall of 0.45. The average precision of the 20,000 weight curve is 0.825. Conversely, the 90,000 weight curve produces maximum recall values of 0.81, but has a corresponding precision of 0.85. The average precision of the 90,000 weight curve is 0.833.

For a balanced approach, we can see that 20,000, 60,000, 80,000, and 90,000 weights produce precision values of approximately 0.90 and recall values of approximately 0.78 at a prediction threshold value of 0.99.

The secondary dataset results are in line with the results, as mentioned earlier. The 20,000 weight curve reveals a maximum precision of 0.98 and recall of 0.45 for a threshold value of 0.99. The 90,000 weight curve shows a maximum recall of 0.81 and precision of 0.85. This shows a nearly identical relation to the above dataset's results.

3.2. Analysis of Results

The questions of which weight to use in the final model will be dependent on the inference dataset that will be seen. As the inference dataset moves further away from our training domain, we would use a less-trained model to have a greater generalisation property.

Given our current test set, our results indicate that further training may not produce any statistically significant benefits; this idea is backed by a calculated standard deviation of 0.047 for precision and 0.067 for recall across all reported weights.

We can also see that model precision and recall values seem to be oscillating as the weights increase. This might be indicative of a local or a global minima within our inherent model distribution function. If this is, in fact, a local minima, further training with a larger learning rate may help escape this issue. Any further model performance increase will need to stem from training, tuning, or dataset augmentation. If this is due to a global minima, then overall changes, such as an expanded or augmented dataset, or a new model, would need to be implemented in order to provide a tangible performance difference.

The values of precision and recall are not unexpected, considering how the domain distance of the training, validation, and testing sets are small; and how YOLOV3-608 reported a mean average precision score of 0.579 when tested on the MS-COCO dataset. Since we are focusing on a single object, as opposed to the eighty classes in the MS-COCO dataset, we can expect to see an increase in average precision over the reported results.

Our secondary dataset produced nearly identical precision-recall curves to the initial dataset. The difference between the two are not statistically significant and can be attributed to the stochastic nature of training the model and the differences in the dataset. We can safely conclude that the additions of vegetation and building models did not significantly contribute to the performance of our model.

The performance of the model indicates that classification on images collected from a SNE can be used to provide accurate detection of terrain objects. This, in turn, provides a feasibility confirmation for applications reliant on the object detection of these terrain objects. Our reported level of recall is reasonable for applications that require total coverage of objects of

interest. Also, the level of precision supports applications that require exact detections.

4. DISCUSSION

The goal of this work was to examine the potential capabilities of deep learning object detection on the terrain domain. The object class selected for the detection task is one that may prove difficult due to its similarity to surrounding terrain and rocky surfaces. As we can see in Figure 6, our trained model may not always correctly identify the berm object.

As we can see from our analysis, our model performance can provide fairly accurate detections through the use of a simple data collection session. Given the success and difficulty of this class's detection, we can expect to expand to other more distinctive classes with reasonable success. This work is intended to lay the groundwork for future terrain generation applications that can scale to hundreds, if not thousands, of classes with little additional manual effort and with competitive performance, regardless of need.

Figure 6: Example of False Positive Detection

5. CONCLUSION

Training simulations may benefit from realistic SNEs; yet effectively and efficiently creating these environments to match the RW has been a challenge. In this paper, a current terrain import limitation of the Unity game engine was discussed, as well as an in-progress solution. Additionally, this paper used a deep learning algorithm to support the automated re-creation, or reconstruction, of RW environments. Ultimately, the results suggest that the methodology of applying a combination of photogrammetry and 3D scanning can generate a high-fidelity SNE that can then be used to accurately train a deep learning object detection model to populate the said environment with detected classes. This pipeline can be scaled to hold several more classes and cut the cost of labour for high-fidelity SNE generation.

5.1. Limitations

The largest hurdle with the present methodology is the initial model training. To create the final model, one must first train the model with classes of interest. This requires an initial identification and data-capture process in the SNE. After the model is trained on the captured data, the model can then be used to infer on a production dataset. This can be mitigated and pre-trained for common terrain objects, such as trees, flowers, and berms; but more specialised classes may require additional training.

The training process usually requires a large number of images per classes for accurate detection. This process can also be reduced by training on top of our pre-trained model, and by data augmentation techniques, such as rotations, which can be integrated into the data capture session. Future research into other one-shot learning techniques can further reduce the impact of this issue.

After the model has been trained and provided detections, the issue of the dataset's domain can arise. For example, if we train based on berms in a grassy field, and then infer on berms in a snowfield, we can expect a decrease in performance. This is an issue intrinsic to any model: the model will predict based only on what it has seen. The main solution to this issue is to provide a continuous training pipeline to teach the model whenever it encounters new data, predicts false positives, or predicts false negatives. Rough estimation techniques and dataset tools can help human annotators identify these issues. Future research into adversarial networks or actor-critic networks can potentially lead to solutions that ease or replace the human labour of this issue.

Consideration of ill-intentioned individuals in the military domain is a requirement when researching technologies that will be heavily relied on. The act of intentionally fooling a neural network is an active area of research and growing security concerns. YOLOV2 was shown to be susceptible to an adversarial attack (Thys, Van Ranst, and Goedemé 2019). These sorts of attacks are difficult to predict and respond to. This problem is a special case of a model receiving never-before-seen data and not predicting the correct response. The solution to this has been to provide data of the false positives and false negatives or to tune the model's level of discrimination through the allowable thresholding.

5.2. Future Research

As a next step, choosing the appropriate form of a machine-learning algorithm will be valuable; one variant of machine learning to investigate is one-shot learning. The one-shot learning technique in machine learning is used to quickly train a model on a new class using only a few training images. The difficulty of this method comes from the model's inability to reflect on past-learnt classes and find similarities to new classes, in order to rapidly learn. The adoption and research of this technique can help scale models to new classes greatly.

The present experiment focused on testing the training and classification of one type of feature: a berm. Given the diversity of RW environments, a larger range of class types should be considered in future deep learning train-and-test paradigms.

In conjunction, if game engines in military training are to become accepted as valuable in terrain reconstruction, building advantage profiles per each system (e.g., Unity, Unreal, and CryEngine) is desired: defining which game engines are most effective and efficient at building SNEs should be considered.

Further, IST is developing a method to allow direct import of 3D mesh files (e.g., OBJ and FBX) to be losslessly reinterpreted as native Unity terrain (see Figure 7). This solution eliminates the need to use a 16-bit depth heightmap. While not fully mature, this solution deserves further research.

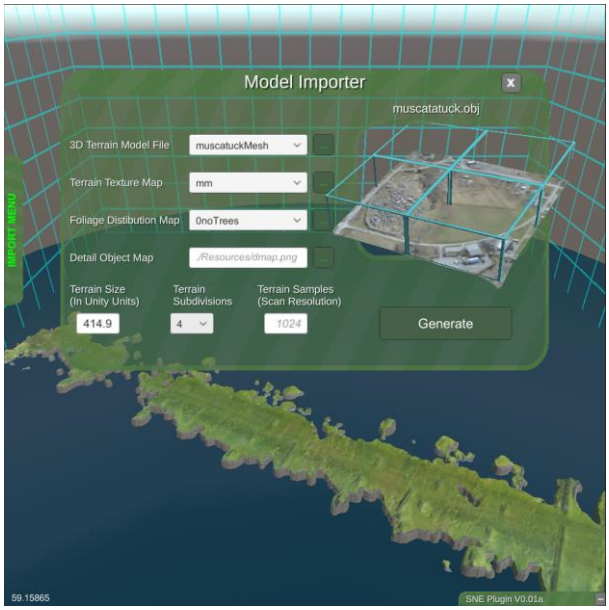

Figure 7: Screenshot of the Unity Terrain Importer

ACKNOWLEDGMENTS

This research was sponsored by Latika (Bonnie) Eifert of the U.S. Army Futures Command, Combat Capabilities Development Command-Soldier Center Simulation and Training Technologies Center (STTC). However, the views, findings, and conclusions contained in this presentation are solely those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

REFERENCES

Baker T., 2018. VBS3 Resource. Milgaming website. Available from: https://milgaming.army.mil/VBS3/files/ResourceList.aspx [accessed 14 Mar 2019].

Chen Y., Jiang H., Li C., Jia X., Ghamisi P. , 2016. Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. IEEE Transactions on Geoscience and Remote Sensing, 54 (10), 6232-6251.

Kemker R., Salvaggio C., Kanan, C., 2018. Algorithms for semantic segmentation of multispectral remote sensing imagery using deep learning. ISPRS Journal of Photogrammetry and Remote Sensing, 145, 60-77.

Micro Aerial Projects L.L.C., n. d. V-map AIR 20Hz GNSS receiver technical specifications. Micro Aerial Projects L.L.C.

PEO STRI, n.d. Virtual targets. PEO STRI website. Available from: https://www.peostri.army.mil/virtual-targets [accessed 14 May 2019]

Reed D., Thomas T., Eifert L., Reynolds S., Hurter J., Tucker F., 2018. Leveraging virtual environments to train a deep learning algorithm. Proceedings of the 17th International Conference on Modeling and Applied Simulation (MAS 2018), September 17-19, Budapest (Hungary).

Spicer R., McAlinden R., Conover D., 2016. Producing usable simulation terrain data from UAS-collected imagery. Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), pp. 1-13.

Tchou C., 2018. 2018.3 terrain update: Getting started. Unity Blog. Available from: https://blogs.unity3d.com/2018/10/10/2018-3-terrain-update-getting-started/ [accessed 10 May 2019]

Thys, S., Van Ranst, W., & Goedemé, T. 2019. Fooling automated surveillance cameras: adversarial patches to attack person detection. arXiv preprint arXiv:1904.08653.

Zhang L., Li Z., Li A., Liu F., 2018. Large-scale urban point cloud labeling and reconstruction. ISPRS Journal of Photogrammetry and Remote Sensing, 138, 86-100.

AUTHORS BIOGRAPHY

Dean Reed (B.S. in Computer Science, UCF 2000) is a Senior Associate for Simulation with the Institute for Simulation and Training (IST) of the University of Central Florida. Mr Reed is a veteran of the U.S. Army and leads a team of developers at IST. He has worked on a vast array of projects under the auspices of the University, including NASA Vision Spaceport. He is currently managing team efforts directed at evolving future training ranges on behalf of the U.S. Army.

Troyle Thomas (M.S. in Computer Science, UCF 2018) is an Assistant in Simulation with the Institute for Simulation and Training (IST) of the University of Central Florida. His research interests include computer vision, natural language processing and machine learning applications, with particular attention to unsupervised learning methods. Mr Thomas has been at the Institute for Simulation and Training since 2017. His primary responsibilities involve the research of artificial intelligence techniques, with a primary focus on deep learning, and their application to the virtual

environment, embedded environments and interactive training.

Shane Reynolds is a graduate of UCF in Digital Media. He has specialised in compelling 3D content development and mobile game engine development for over ten years. Mr Reynolds is a veteran of the U.S. Air Force. Shane is a Research Associate at the Institute for Simulation and Training where he has been a faculty member since 2008. His primary activities involve research and integration of modern technologies to train dismounted Soldiers at the squad level. Currently, his focal areas are technologies involving virtual reality, augmented reality, and photogrammetry.

Jonathan Hurter is a Research Assistant at the University of Central Florida's (UCF's) Institute for Simulation and Training (IST). Holding a Master's degree in Modeling & Simulation from UCF, Jonathan has worked with human-based research topics, including the relation of avatars with performance, the usability of virtual reality systems, and the effects of instructional strategies for signal detection. His efforts fall under instructional design and technical communication, mainly.

Latika (Bonnie) Eifert (M.S. in Computer Engineering, UCF 2003) is a Science and Technology Manager at the U.S. Army Research, Development and Engineering Command (RDECOM), Army Research Laboratory, Human Research Engineering Directorate, Simulation and Training Technology Center (ARL-HRED ATSD) located in Orlando, Florida. Ms Eifert manages several projects associated with simulation and training. She is also supporting the Defense Advanced Research Project Agency (DARPA) by managing research program efforts.