Artificial Intelligence Supported Site Mapping for Building Pop-Up Habitats

Atousa Aslaminezhad, Henriette Bier, Arwin Hidding, and Giuseppe Calabrese

Abstract Building pop-up habitats in extreme weather conditions such as deserts requires preliminary contextual, i.e., site studies. Since the site's condition is constantly changing due to sand relocation induced by wind, a rapid mapping solution is proposed. This is implemented by generating a 3D mesh model of the site with the help of a visual workflow and advanced computational design methods to implement in-situ 3D printing of habitats. This paper presents an integrated approach utilizing Computer Vision (CV), Deep Learning (DL), and generative design tools like Grasshopper. By harnessing the potential of Convolutional Neural Networks (CNNs), a robust framework is developed to recognize complex desert terrain features, independent of solar orientation and camera positioning. The methodology employs a state-of-the-art CNN customized for detecting features in desert settings. This is further enhanced by using Grasshopper to systematically generate a diverse dataset that enriches the model's learning process. The resulting model efficiently extracts precise 3D meshes from 2D images, optimizing site mapping and integrating habitat printing workflows. This automated approach offers an effective solution for habitat construction in challenging environments, showcasing real-time processing.

A. Aslaminezhad (✉)
University of Antwerp, Antwerp, Belgium
e-mail: a.aslaminezhad@hw.ac.uk

Heriot-Watt University, Dubai, United Arab Emirates

H. Bier · A. Hidding
Delft University of Technology, Delft, Netherlands
e-mail: h.h.bier@tudelft.nl

A. Hidding
e-mail: a.j.hidding@tudelft.nl

H. Bier
University of Sydney, Sydney, Australia

G. Calabrese
International Research School of Planetary Sciences, Università d'Annunzio, Pescara, Italy
e-mail: giuseppe.calabrese1@unich.it

V. Bhateja et al. (eds.), *Evolution in Computational Intelligence*, Smart Innovation, Systems and Technologies 436, https://doi.org/10.1007/978-981-96-2124-8_10

1 Introduction

Recent advancements in Artificial Intelligence (AI) and in particular Computer Vision (CV) for three-dimensional mesh generation from images offer new opportunities for site layout mapping [1]. This paper explores the capabilities of CV to extract 3D meshes from a single photograph in real-time in the desert.

By applying Convolutional Neural Networks (CNNs), the presented study seeks to open novel prospects in constructing pop-up habitats in the desert through robotic 3D printing. Given the complexity involved in the dataset collection and prohibitive costs, this research turns to computer-generated datasets as significant resources for AI model training.

1.1 State-of-the-Art Terrain Mapping Techniques

Terrain mapping is fundamental across various domains, from urban planning to environmental monitoring. Over time, several methods [2–4] have emerged to accurately depict the intricacies of different landscapes. Below, outlined are some cutting-edge terrain mapping techniques currently utilized:

From Lidar technology employing laser pulses to photogrammetry extracting 3D information from 2D images captured from varied perspectives and satellite remote sensing using multi- or hyperspectral sensors, all generate valuable data for terrain mapping.

Specialized algorithms have been designed to depict specific attributes of desert landscapes, including features like sand dunes, rocky formations, and sparse coverage of vegetation employing techniques such as cellular noise patterns and Perlin noise functions to enable the accurate simulation of the terrain. Furthermore, Voronoi diagrams which divide space into regions by the proximity to specific points, are employed in modeling desert terrains. They aid in generating realistic patterns resembling the distribution of sand dunes and rocky objects found in desert landscapes.

More advanced methods involving Convolutional Neural Networks (CNNs) that are trained on extensive datasets, can automatically recognize and categorize terrain components like sand dunes, rocky surfaces, and vegetative cover, facilitating terrain mapping and evaluation.

In this context, demonstrating such approaches is effective since they provide real-time modeling of terrain surfaces based on different sources of sensor data such as Lidar scans or radar images. The use of effective algorithms along with the implementation of parallel processes helps to generate 3D models immediately, making it possible to navigate autonomously and respond in deserts. Meanwhile, the challenging dynamic landscapes (e.g., wind pattern shifts or erosion) call for the development of advanced land-forming technology. Such techniques maintain the

density of sampling dynamically, by adjusting the complexity of the terrain; thus the models are closer to the actual conditions of the terrain as it changes with time.

Furthermore, some research highlighted that this approach would be compatible with the application of CV methodologies and in-situ 3D printing for the sustainable implementation of extraterrestrial habitats in the framework of lava tubes by obtaining depth maps [5].

In summary, these advanced terrain mapping approaches offer multidimensional techniques to contour the terrain, process data, and generate images for multiple uses. Continuous progress in sensor technology and data processing are the driving forces that allow refining and widening the use of terrain mapping capabilities.

1.2 Contribution

This study provides a different approach for generating 3D meshes from 2D single-shot images of desert landscapes and demonstrates the potential of the synergy between Computer Vision (CV), Deep Learning (DL), and generative design tools such as Grasshopper. By connecting DL to Convolutional Neural Networks (CNNs), a robust framework is developed that can recognize and interpret complex desert terrain features for 3D mapping without the need for any interventions such as solar orientation and camera positioning.

This approach, which is in the domain of CV and DL, can be especially useful in the fields of architecture, urban design, and urban planning. The proposed model is developed to provide an efficient and cost-effective approach to construction in extreme environments.

2 Approach and Methodology

The methodology of the presented study involves the development of a robust AI model that can effectively understand, interpret, and rebuild 3D representations of desert sites based on 2D images. This implies the development of numerous solutions to deal with what the desert environment offers while considering what is required for robotic construction. To train the AI model, a dataset of computer-generated 2D images depicting different desert site maps is used. The data validation process is discussed in the following section, during which the accuracy and loss metrics are evaluated. In the most ideal case, the closer the accuracy value is to 1 and the loss value to 0, the better performance of the AI model is expected, and the anticipation is to predict the 3D mesh of the site plan.

2.1 Dataset Generation

Due to a lack of relevant practical data collection as well as logistical challenges related to collecting a high-quality real-world dataset, a synthetic dataset is made using computer simulations. This dataset simulates desert environments with a variety of perspectives for images spanning across the entire spectrum of lighting, such as harsh direct sunlight, subtle and soft shadows, diffused light, and overcast skies.

In this approach, all the parameters that influence the generation of certain datasets are changed for every trial. Hence, every desert model holds one visual source, and separate seeds are used for each model, resulting in a completely random generation of the algorithm parameters.

Each image in the dataset is paired with explicit three-dimensional mesh coordinates developed in the AI model, providing the ground truth needed for effective training.

The data generation process relies on the capabilities of the Grasshopper software which not only eliminates the difficulties of the physical data collection process but also meets the standards to provide a set of images with uniform conditions at high volume demand. This approach is structured in such a way that the data that goes into the training of the CNNs is patterned to closely resemble the complexities of the real-world scenarios they are going to be deployed to. The following steps elaborate on the dataset generation in detail:

Geometric Simulation. The high number of samples necessitates the automatic generation, rendering, and classification of 3D models that are performed in Grasshopper. The algorithm extracts the lines of the dunes with the aid of data generated by the Tundra plugin. Tundra uses the Perlin function, and it takes random parameters based on mathematical relationships. Furthermore, it builds the terrain form of the targeted geographical area via the same function. Moreover, the wind effect on the generated mesh is included. This function is utilized to make the computer-generated mesh correspond to reality. Two generative algorithms were developed to simulate the desert environment:

Cellular Noise Algorithm. This algorithm is built on Worley noise [6] allowing for the creation of Voronoi patterns that imitate the natural way sand, and rocks are distributed (Fig. 1).

Generative Guide Curves Algorithm. This algorithm uses an improved Perlin noise function to generate the guide curves [7] that are involved in the formation of the dune patterns and the site topography (Figs. 2 and 3).

A selection between the two algorithms resulted in a Perlin noise output which is superior in terms of the geometrical diversity and the higher similarity to terrain environment characteristics. Additionally, the Bezier function has also been adapted to simulate the cross-section of dunes.

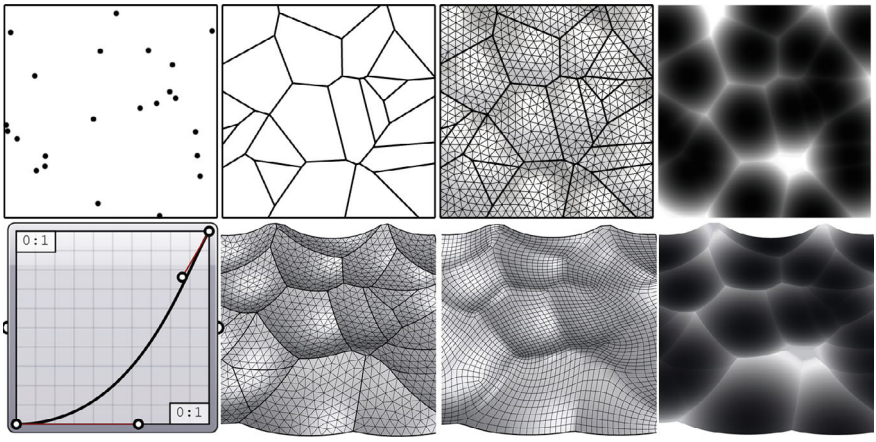

Fig. 1 Modeling steps of Cellular Noise Algorithm: Generating stochastic points (*top left*), Creating Voronoi cells (*top left second row*), Triangulation (*top right second row*), Generated cellular noise map (*top right*), Modifying dunes shape by Bezier graph (*bottom left*), Creating mesh (*bottom left second row*), Smoothening mesh (*bottom right second row*), Depth map (*bottom right*)

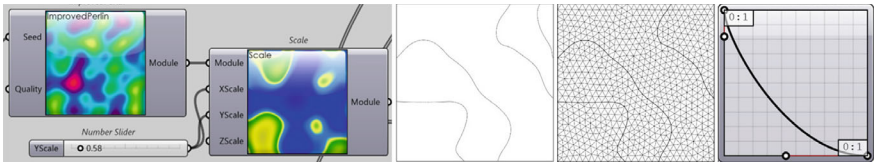

Fig. 2 Step-by-Step Process of Generative Guide Curves for finding random Dune shapes Using Perlin Noise and Bezier Graphs. Defining dune patterns by using the Perlin Noise function with the help of the Tundra plugin [8], and defining wind pattern-X direction (*leftmost*). Extracting dune curves (*middle left*), Surface triangulation for elevation (*middle right*), Modifying dunes shape by Bezier graph; As the mesh vertices get closer to the extracted lines, their height increases according to the Bezier curve pattern (*rightmost*)

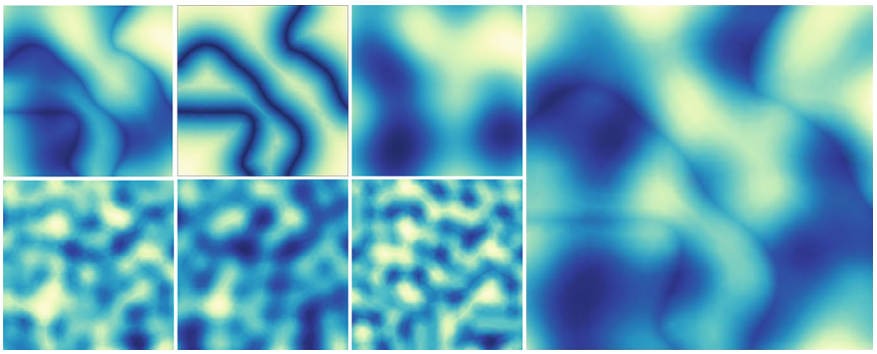

Fig. 3 Example of a pattern produced for dunes. Dune + Terrain (*top left*), Dune (*top middle*), Terrain (*top right*), Wind X + Y (*bottom left*), Wind X (*bottom middle*), Wind Y (*bottom right*), Combination of all patterns (*rightmost*)

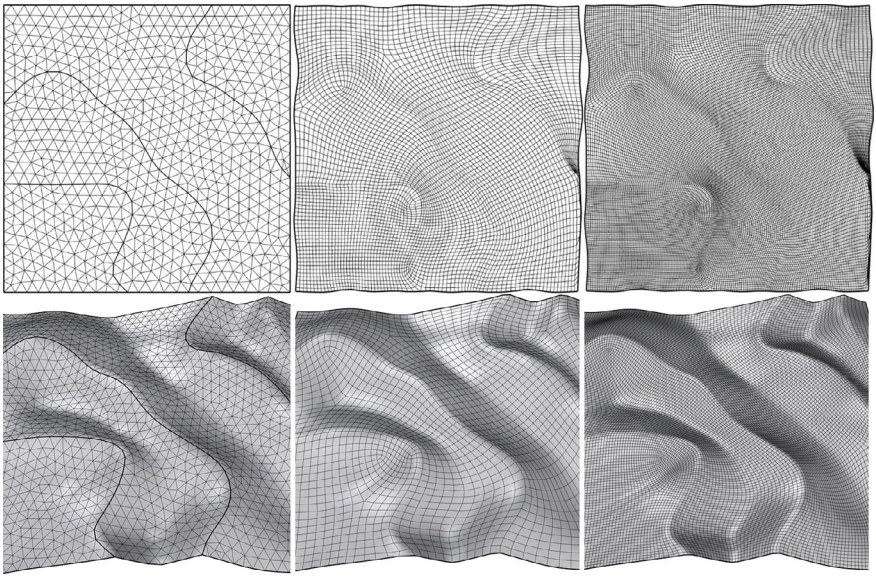

Fig. 4 Geometric prototyping of mesh. Creating mesh (*left*), Creating mesh with sharp dune curves (*middle*), Creating mesh with soft dune curves (*right*)

Geometric Prototyping. To cover the diverse geometrical typologies of dunes, two distinct algorithmic strategies for mesh generation as part of prototyping were investigated (Fig. 4).

Sharp Dune Curves. In this approach, the meshes are designed with protruding or more pronounced dune lines to replicate the sharp features observed in certain desert landscapes.

Soft Dune Curves. This method is aimed to form meshes on which the dunes' slopes are soft and grade to each other which is in line with more gradual slopes found in some desert regions.

Object Scattering. Dubai's desert was selected as the context of this research. Based on the observations made, plants and small stones exist in the landscape which need to be considered in the digital 3D simulations for improved AI model training. To allow all this, a scattering algorithm was used to distribute planes randomly on the original mesh. Five different types of plants and rocks were randomized in different sizes, numbers, and orders across the mesh. The sizes, quantities, rotations, and distribution patterns were controlled by this algorithm (Fig. 5).

Visualization. The V-Ray plugin was employed due to its advanced rendering capabilities to simulate realistic lighting and material properties, and its full integration as a plugin in Grasshopper allows automated dataset generation while giving realistic renderings.

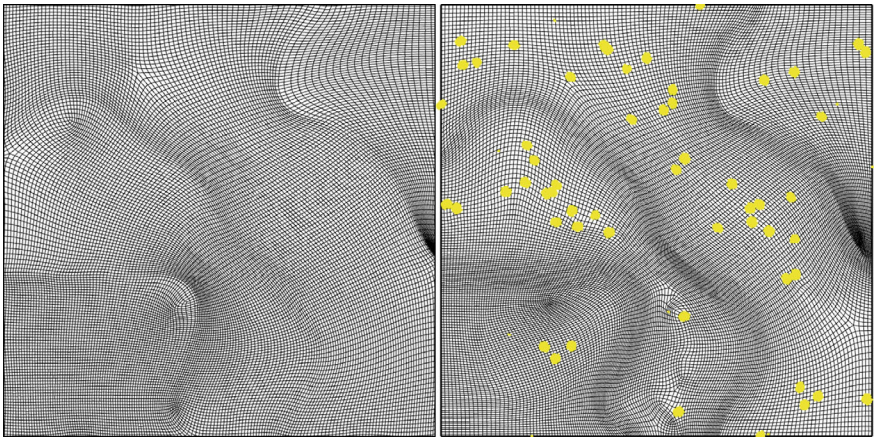

Fig. 5 Object scattering on terrain. Site plan mesh (*left*), Scattered objects (*right*)

The material for dataset production was sourced by considering ten different types of materials found in the desert region of Dubai. The images were rendered with a resolution of 500 × 500 pixels, using an RGB color channel . Figure 6 shows selected images of the dataset.

Initially, the dataset was set at 10,000 images. However, 10% of the images were either corrupted or misclassified which led to the growth of the initial dataset size to 11,000 renderings. All images were automatically V-Ray rendered and saved in the transition from 0 to 11,000. At the same time, the Z coordinates of each vertex from the studied mesh were saved as a separate list in a Comma-separated Values (CSV) file. The corrupted and incorrect data was manually deleted from the dataset.

Camera Settings. In this model, an algorithm is employed to determine the positions of the sun and the camera. It creates two points between two spheres with the target point located at the center of the site plan and on the ground surface, encompassing

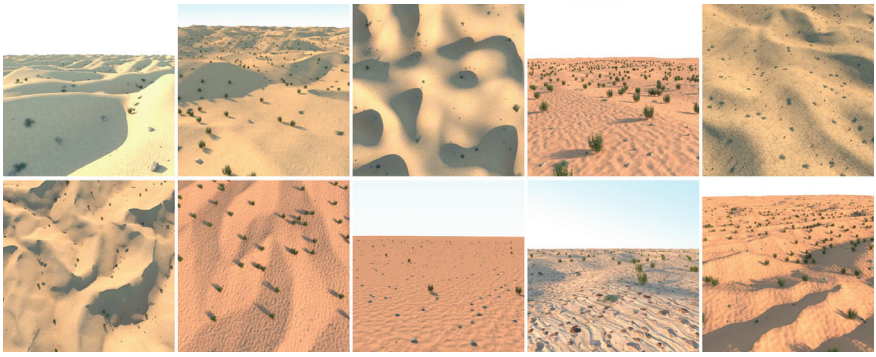

Fig. 6 Selected samples from the generated dataset

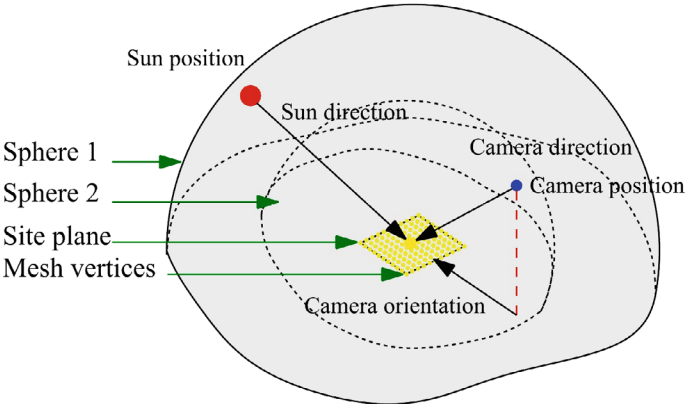

Fig. 7 Photography and camera placement model

all points. Some parameters such as the direction of sun and light, the placement of the camera, site plan, elevation, overall topography, initial noise, and vegetation coverage will have a significant influence on the model. The Field of View (FOV) of Apple's iPhone camera was used, with a value of 54° [9].

The distance between the camera and the center of the mesh remains unchanged. Consequently, because of the randomly chosen shooting angle, the camera is perpetually oriented towards a sphere with a radius of 18.5 m, centered on the mesh that is being analyzed (Fig. 7). For the mesh to be within the camera's FOV and be fully visible, 18.5 m was chosen as the distance between the camera and the examined mesh.

Sun Position and Intensity. The sun's position is generated on a large sphere with a radius equal to the mesh's overall size. It is designed using a day-and-night system always to ensure the presence of the sun avoiding nighttime hours. For obtaining this light intensity and simulating various weather conditions, a sigmoid function that maps input values to a range between 0 and 1 was employed in the algorithm.

2.2 Preparing Dataset for the AI Model

The dataset images are first amended via Photoshop in batches based on some parameters such as hue, saturation, contrast, and histogram, and then the outlier data are deleted via a developed Python algorithm. Additionally, the images with modeling or visual errors are removed from the dataset. The 10,000 final images of the desert landscape generated in the preceding step have been standardized as pictures in JPG format. Besides that, the vertices of each mesh are simultaneously listed and positioned in the Z coordinates, kept in a CSV file format to be used for building the AI model. All images are standardized equally as they are recorded by the same camera

settings with a common output size from Grasshopper. This can help improve the model's overall performance.

The study produced a primary and a secondary mesh. The primary mesh is the result of the Generative Guide Curves (GGC) algorithm which is the comprehensive and seamless reproduction of the natural features of the desert context. The secondary mesh, which is derived from the primary mesh, is a 10×10 square mesh grid whose vertices are spaced 1m apart and oriented in a way that its center aligns with the center of the image. This gives the camera real-time updates from the mesh for any further instant processes.

3 Deep Convolutional Neural Networks for 3D Mesh Prediction from Site Images

The core of the methodology is the usage of the CNN architecture which is tailored to allow Z coordinates extraction of 3D mesh vertices from a single 2D site plan image. CNNs have shown high efficiency in performing Computer Vision tasks like image classification, image understanding, and image segmentation to object detection [10]. These models can learn complex image feature representations through training, thereby making them appropriate for this study. The CNN architecture, training, and evaluation process involves the following steps.

3.1 Data Processing

In this study, the AI model is implementing the data that it has already generated for training purposes. The image's pixel values serve as input (X_train), with the mesh Z coordinates as output (y train). The parameters of the model were adjusted during the training to minimize the difference in the coordinates between the actual and the predicted meshes for increased accuracy.

The methodology uses the AI model to detect the required mesh and to navigate robotic construction in extreme environments in real-time.

The code establishes essential libraries such as Pandas for data manipulation, Autogyro for numerical calculations, Tensorflow for Deep Learning (DL), and Python Imaging Library (PIL) for image processing [11]. File references are in the directory of 3D site plan images of deserts as well as the CSV file for storing 3D mesh Z coordinates.

3.2 Loading Images and Coordinate Extraction

The code iterates through each row of the CSV file:

In this step, the code receives the images and turns them into a float-point NumPy array. This transformation is made to promote efficient information processing at the neural network's internal level. The code then gains the 3D mesh Z coordinates of the image in the current row through the CSV file. This extracted geography information is added to the "coords" list. Additionally, the processed images are saved into another list called "images".

The code arranges the images into 10,000 for its algorithm. Each cell of array contains a list of three values to represent the RGB channels of each pixel, which is used as input data. Hence, the pixel values list of the images are referred to as X_ train, and the coordinates of the site plan list are referred to as y_train.

3.3 Data Splitting

The lists representing these arrays get converted into Numpy arrays, which will be referred to as X and y. X represents the list of input images and y represents the corresponding 3D vertices, i.e., Z coordinates of the mesh. Training and testing dataset partitioning is achieved via sklearn's train_test_split function with test_size of 0.2 which includes 20% of the data being tested and 80% being trained. The splatted datasets are X_train, X_test, y_train, and y_test. The splitting allows the model to demonstrate its performance for any unseen datasets while being tested thus estimating the data generalizability.

3.4 Data Normalization

A Keras's ImageDataGenerator object helps to perform data standardization by normalizing the image data. This operation is quite valuable in improving the model performance and training process. The idea of normalization focuses on pixelation by scaling and zeroing down the values. However, the range is factored in between 0 to 1. Hence, the variations impact in image illumination and intensity are minimized which provides a smoother learning process for CNN.

3.5 CNNs Architecture

Convolutional Neural Networks (CNNs) as one of the best networks for image recognition are used. CNNs which were trained on the synthetic dataset to identify and

predict 3D forms of the terrain from the 2D images, were carefully designed to extract macro features of the desert. This led to high and precise 3D mesh modeling.

Demonstrating the core of the system as a deep CNN model, made with a Sequential API from Keras. API makes provision for sequential deposition of layers which in turn is a platform for the creation of complex neural network architectures.

The CNN architecture is thoroughly designed to process the input images and extract the essential features that indicate the 3D forms in the site plans. This architecture consists of several layers, each with a specific function:

Input Layer. The first layer describes the shape of each input image data. This layer provides information about the size and the number of channels e.g., RGB in the model.

Convolutional Layers. The model consists of two convolutional layers of 32 and 64 filters with 3×3 size, respectively. Convolutional layers are the building blocks of CNNs. They perform an operation of convolution, extracting spatial features from the raw data and input images [12]. Every filter detects a small detail, such as an edge, texture, or pattern present in the image. The size of these filters plays an important role in high-resolution details in the network. After every convolutional operation, a function such as Rectified Linear Unit (ReLU) is involved to generate non-linearities in the model. This allows the model to learn complex relationships between the features.

Pooling Layers. Two pooling layers with a pooling size of 2×2 are used after each convolutional layer. The filtering layers are used for pooling the feature maps and reducing the resolution, thus retaining only the most important features while providing invariance to features' position in the input image as well as reducing the computational load for the next layers. This approach assists in keeping the model from being complex and in avoidance of overfitting. Figure 8 shows selected pooling feature extraction samples.

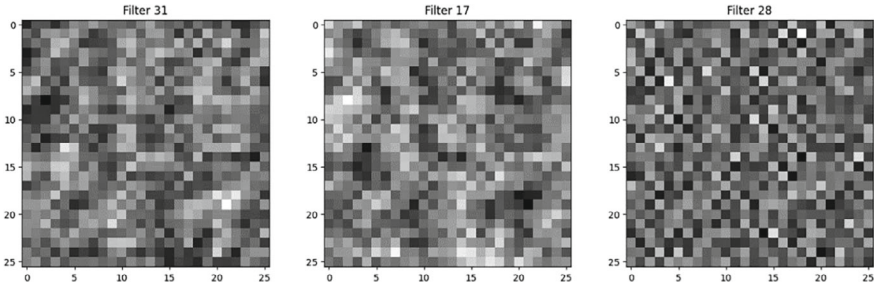

Fig. 8 Feature maps extracted from CNN layers: visualization after applying filters and pooling operations

Flatten Layer. This layer converts the two-dimensional feature maps generated by the convolutional and pooling layers to a one-dimensional vector. This gives a lower-dimension representation of the input data making it operational for the next layers that are only working with one-dimensional data.

Dense Layers. The fully connected layers integrate the locally learned features thereafter to make universal predictions about the content of the images. In the context of 3D mesh prediction, these layers produce coordinates that help build the structure of the meshes. The fully connected dense layers that are employed in the later stages of the network execute a matrix multiplication between the flattened feature vector and the weights assigned to each neuron. The first two dense layers are followed by each other with 100 neurons and the ReLU activation function. As a result, the model can further improve more complex relationships between the extracted features. The last dense layer has 100 neurons and softmax activation function while its output dimension corresponds to the size of the Z coordinates for each vertex of the 3D mesh.

4 Model Training and Evaluation

4.1 Model Compilation

The model.compile function configures the model for training by specifying the following:

Loss Function. In terms of 3D mesh prediction, which is a regression task, a usual choice is the MSE (Mean Squared Error) (1) measure to assess the squared mean differences of the predicted and given accurate values for training data. Reduction of the MSE via training leads the model to be as precise as it can, while making predictions that are as close to the true values as possible. This, in turn, investigates the model to reconfigure its inner parameters during the training process (weights and biases) to minimize the differences.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{1}$$

Optimizer. The Adaptive Moment Estimation optimizer is a learning algorithm that iteratively improves the model by repeated weight and bias adjustment of parameters according to the calculated loss at each step of the training process. It strives to select the weights that minimize the amount of loss among all the training instances over all the sets of training datasets. The Adam optimizer was used in its default mode in the Tensorflow library with a learning rate of 0.001.

Evaluation Metrics. It is the metrics argument that enables the observation of the extra parameters while the trainers and evaluators are working without the need for a third party. The mean absolute error (MAE) (2) metrics are adapted here to measure consistently the average absolute discrepancy between predicted and real coordinates. This gives a detailed look at how the model performed compared to the loss function.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right| \tag{2}$$

4.2 Model Training and Evaluation Criteria

Training a convolutional neural network (CNN) requires many steps which include adjusting the network's weights according to data, which is needed for the model to learn. During this process, i.e., the backpropagation, the goal is to reduce the value of the loss function—this step plays an important role in guiding the training process. The prime issues are the choice of loss function and the evaluation metrics. Moreover, the training process includes a hyperparameter setting, which implies the process of adjusting parameters like learning rate, batch size, and the number of epochs. These hyperparameters inevitably make a profound impact on the training process which eventually may contribute to the final performance quality of the model.

Model Training Process. The model.fit function is the main starting point of the training process. During training, the function iterates through the training data (X_ train, y_train) in batches and performs the following steps within each epoch:

Forward Pass. The input image of the batch sample in the current processing is carried out by the network, producing a predicted outcome for the corresponding 3D coordinate mesh.

Loss calculation. The loss function evaluates the gaps between the model's predicted coordinates and the actual coordinates.

Backward Pass. An increased loss is then propagated to the network using the back-propagation method, and the model thereby can modify its weights and biases to minimize the loss in subsequent passes.

Weight Update. The optimizer updates the weights and the bias of the model depending on the gradients computed during the backward pass.

Model Evaluation Process. The model's performance is periodically evaluated on a special set of validation data during training to verify its generalization capabilities and avoid overtraining. This means that the model can be trained with the data which is not seen by the model during training and then it can be used to test the predictions.

The model.evaluate function will give an overview of the model's performance on the unseen testing data by using (X_test, y_test). The respective loss values (test_loss) and MAE (test_mae) are calculated and printed to provide insights into the estimation of how well the model is performing against an unknown dataset. Ideally, both values should be as minimum as possible.

4.3 Model Optimization

The model is trained based on the evaluation results. Appropriate modifications are made to the architecture or the training parameters and hyperparameters towards the model's enhancement.

4.4 Model Saving

The model.save function is used to save the optimized and trained model. This makes the model capable of keeping the knowledge it learns for future use.

4.5 Model Loading

The load_model function is used to load the previously saved model from the given path. Hence, the trained model can be used for the prediction of new images.

5 Prediction of New Images

To generate 3D mesh coordinates on a new image, the trained model undergoes a systematic process: loading the image and then preprocessing it to the model input template. Following this, the model.predict function is introduced to compute the mesh coordinates which are subsequently organized as an output list. Finally, the data is shown as a mesh in Grasshopper. Figure 9 shows three ideal predicted images. Flags represent predicted points.

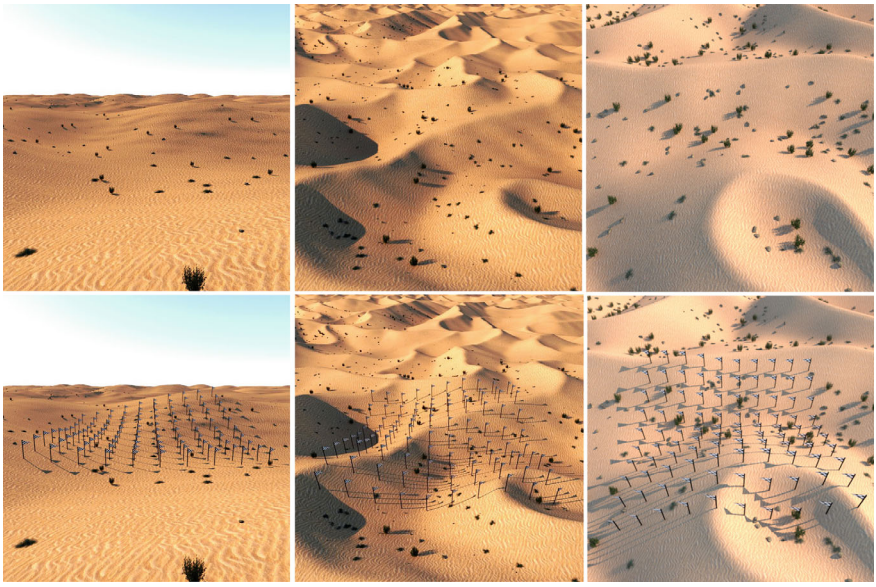

Fig. 9 The ideal predicted samples. X_Predicts (*top row*), y_Predicts (*bottom row*)

6 Results and Discussion

The algorithm ran several times to evaluate the model's performance based on its accuracy and loss metrics on both training and validation datasets. The evaluation is based on the training epochs and the experiment results as shown in Figs. 10 and 11. Due to the lack of real data, Human Evaluation and Statistical Comparison (Pixel-level comparison) methods are used to validate the dataset. The Human Evaluation includes Expert evaluation and Perceptual studies that generate acceptable similarities between them by examining the computer-generated dataset. In the Statistical Comparison, 400 generated images were compared against 20 real desert images using a Python algorithm and the results show that computer-generated images are similar to the real ones in terms of brightness and variance however, there are some differences in terms of details and diversity. So, to avoid an imbalanced dataset, the V-Ray rendering settings and preprocessing operations need further improvements.

Training Accuracy. The training accuracy has been improved from 0.025 to 0.246. This improvement indicates that the model can learn the characteristics in the training dataset and becomes more accurate in predicting.

Validation Accuracy. The validation accuracy has also increased from 0.015 to 0.194. Although this shows progression, the fact that it is much lower than the dataset accuracy indicates that the model most likely overfits the training data. This occurrence arises when a model prioritizes memorizing training data specifics rather than

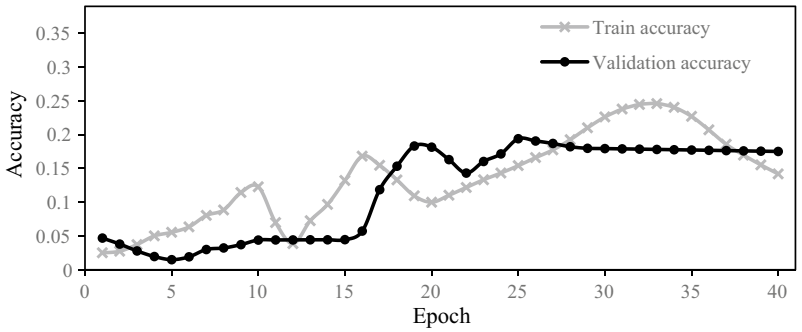

Fig. 10 Training and validation accuracy diagrams within 40 epochs

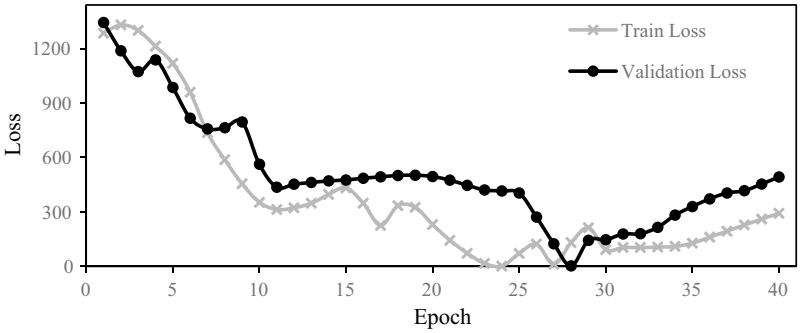

Fig. 11 Training and validation loss diagrams within 40 epochs

learning generalizable patterns. Hence, the model excels in performing on training data.

Figure 11 shows the model's learning curves during the training. The difference between the training and validation accuracy curves shows a potential overfitting.

Training Loss. The training loss has been significantly minimized from 1330.103 to 1.2537. This decrease in loss implies that the model is increasingly learning from each instance in the training data and as a result, its predictive accuracy is improving with each iteration.

Validation Loss. The validation loss has also decreased from 1343.344 to 2.3436. This indicates that the model is improving its precision of prediction on the validation set.

Overall, the model shows exponential growth in the learning curve with increasing accuracies and decreasing losses. Nevertheless, the difference between the performance on training and validation reveals the possibility of overfitting. An enhanced model can be made by adding strategies such as regularization, data augmentation,

and incorporating a complex validation set. Regular tracking of these metrics will be essential for tweaking and enhancing the performance of the model.

Based on the results from the model's output, the AI model is currently not very reliable, and the results cannot be taken as proven. Nevertheless, it is revealed through the assessment that CNN algorithms likely improve the performance detection of the mesh using images, and the results are promising. Since the purpose of mesh detection in this study is for the construction of pop-up habitats, a difference error between an actual and predicted coordinate of a few centimeters is acceptable. For instance, if all points have a prediction difference of 0.05, the loss value would be 0.0025. Therefore, the next step for improving this research is to minimize the loss value that can be further studied.

The inherent difference between the synthetic dataset and the real-world terrains will cause inaccuracies in the predictions. The training of the model should be updated with high-resolution height map scans of real-world terrains, in combination with real photographs, so the model's reliability can be improved. These benchmarking scans represent the true or ground truth surfaces and allow the model's true error to be identified and can therefore be reduced. Because the model has been trained on synthetic datasets, the amount of real-world scanning data is significantly reduced. The error between the true landscape and the predicted meshes needs to be within production tolerances to make the model usable for in-situ production. Since there will always be an irreducible error between the predicted meshes and the true landscape, some flexibility and adaptability need to be built into the (semi)-robotic on-site production process.

7 Conclusion

This study provides an essential point in the scheme of data processing. The successful implementation of real-time 3D mesh detection for in-situ 3D printing in desert terrains exhibits how AI can change the problems of harsh and unpredictable environments.

The creative utilization of CNNs as add-ons and computer simulations has led to great advancements in Computer Vision and robotics.

The study proves that there is a potential for much wider implications of AI in construction than the immediate applications. The knowledge obtained may be used to build on further innovations, facilitating architects to develop habitats in extreme environments where accessibility is challenging. It shows the prospect of employing advanced technologies such as Computer Vision and Deep Learning in the construction of pop-up habitats in extreme environments, i.e., deserts. Using synthetic datasets generated by computer simulations, Convolutional Neural Networks are a promising method for predicting the 3D mesh coordinates from 2D images of desert landscapes. Despite the large amount of progress being made towards the improvement of factors such as the training process along with finding out the accuracy of loss, some challenges like the overfitting that occurs between synthetic and real-world scenarios

during implementation need to be addressed to simplify and enhance the processes for better practical use. Using real-world data such as high-quality height map scans and actual photos, for instance, can improve the training process as well as the consistency of the model in producing results and minimizing inaccuracies. As a result, two major assets needed to aid in this process include resilience and adaptability, both of which help accommodate for unexpected errors between a predicted mesh and the true landscape, ensuring a very usable and practical process for the real world.

Acknowledgements This study was developed by the contribution of researchers from the University of Antwerp, Delft University of Technology, and Università d'Annunzio.

References

1. P.K. Vinodkumar, D. Karabulut, E. Avots, C. Ozcinar, G. Anbarjafari, Deep learning for 3D reconstruction, augmentation, and registration: a review paper. Entropy (Basel, Switzerland) **26**(3), 235 (2024). https://doi.org/10.3390/e26030235
2. O. Olson et al., Visual terrain mapping for mars exploration. Comput. Vis. Image Understanding **105**(1) (2007). https://doi.org/10.1016/j.cviu.2006.08.005
3. M. Pieraccini et al., Terrain mapping by ground-based interferometric radar. IEEE Trans. Geosci. Remote Sens. **39**(10) (2001). https://doi.org/10.1109/36.957280
4. V.L. Mulder et al., The use of remote sensing in soil and terrain mapping—a review. Geoderma **162**(1–2), 1–19 (2011). https://doi.org/10.1016/j.geoderma.2010.12.018
5. G. Calabrese, A. Hidding, H. Bier, C. Engelenburg, S. Kahdemi, A. Aslaminezhad, Computer vision for terrain mapping and 3D printing in-situ of extra/-terrestrial habitats, in *Intelligent Systems Conference (IntelliSys)* (2024)
6. S. Worley, A cellular texture basis function, in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996)
7. A. Jain, A. Sharma, Rajan, Adaptive & multi-resolution procedural infinite terrain generation with diffusion models and Perlin noise, in *Proceedings of the Thirteenth Indian Conference on Computer Vision, Graphics and Image Processing* (2022)
8. (Miroljub), *Tundra.* Food4Rhino (2019, February 16). https://www.food4rhino.com/en/app/tundra
9. FieldOfView, Apple developer documentation (n.d.). Retrieved 2 April 2024, from https://developer.apple.com/documentation/modelio/mdlcamera/1391726-fieldofview
10. M. Egmont-Petersen, D. de Ridder, H. Handels, Image processing with neural networks—a review. Pattern Recogn. **35**(10), 2279–2301 (2002). https://doi.org/10.1016/s0031-3203(01)00178-9
11. W. Ballard, *Hands-On Deep Learning for Images with TensorFlow: Build Intelligent Computer Vision Applications Using TensorFlow and Keras* (Packt Publishing, 2018)
12. S. Bhattacharyya, V. Snasel, A. Ella Hassanien, S. Saha, B. Tripathy, *Deep Learning: Research and Applications* (De Gruyter, Berlin, Boston, 2020). https://doi.org/10.1515/9783110670905