Blending Output from Generative Adversarial Networks to Texture High-Resolution 2D Town Maps for Roleplaying Games

Gianfranco Siracusa
Department of Artificial Intelligence
University of Malta
Msida, Malta
gianfranco.siracusa.00@um.edu.mt

Dylan Seychell
Department of Artificial Intelligence
University of Malta
Msida, Malta
dylan.seychell@ieee.org

Mark Bugeja
Department of Artificial Intelligence
University of Malta
Msida, Malta
mark.bugeja@um.edu.mt

***Abstract*—The recent success of Generative Adversarial Networks (GANs) in image and video applications led to the development of numerous variants specialised for particular tasks, such as conditional GANs for image-to-image translation. In spite of the research done in fine-tuning architectures and applying them to different subjects, techniques still deal with stand-alone images, such as nature scenes, city landmarks, faces and others. The task of producing contiguous colour data – namely adjacent parts of the same image, not textures – has not been attempted before in literature related to generative machine learning techniques. Achieving this feat would allow large images to be processed in smaller parts, hence removing the architectural maximum to the output resolution that can be achieved by the network. Concurrent state-of-the-art architectures for conditional image-to-image translation are in the range of 2k×1k pixels and typically take several days to train on powerful hardware. The proposed contiguous technique, in this case applied on fantasy maps for roleplaying games, can achieve higher resolutions with smaller networks that can be trained faster, within a single day. The technique is capable of maintaining as much quality as allowed by the detail of the semantic layouts provided, even at 4k and higher, but it suffers when detail in these is too sparse. A sample of images produced by the system were shown to survey participants who judged their appeal as 3.49 on a Likert scale of 5, and segmentation analysis reported an average weighted inter-class accuracy score of 0.689 (0.448 unweighted).**

***Index Terms*—conditional generative adversarial networks, image-to-image translation, map texturing, game content creation**

Fig. 1. A full-width window from an 8k×8k map textured using a 512×512 image-to-image-translation network by blending contiguous runs. The result typically retains the quality of a deeper, larger network that is more difficult to train. A seam is visible on the upper right corner where detail is sparse (large area of grass), but none are evident in other areas. Kindly zoom in for more detail. The semantic layout that produced this image was generated procedurally using [1].

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

I. Introduction

Content creators for roleplaying games, particularly tabletop games, are constantly on the lookout for good quality, custom maps to use in their works. Maps put the player in the context of the game world and significantly improve immersion [2] for those who appreciate depth as opposed to casual gameplay.

Creators who cannot draw their own maps can look to some alternatives. Finding maps online is easy and free, but these are harder to adapt to custom requirements and may run into copyright concerns when publishing. Professional artists may be commissioned to overcome these limitations. Paid map-making software exists (such as [3] and [4]) that assists in the making high-quality custom maps using a sprite library. Some of these may generate procedural layouts but still build the resulting image by assigning pre-made object textures from the artist-drawn library. In fact, generating new textures for map objects is a current limitation in procedural map generation, as demonstrated also by the fact that fully procedural generators manage to create excellent-quality layouts with minimal user effort, but result in either plain colour output (such as [1]) or simplistic shape patterns (such as [5]).

The previously mentioned disadvantage of the otherwise powerful procedural techniques served as a motivation to develop a system to aid in creating good quality, original maps, with minimum design and drawing effort, that does not rely on a library of hand-drawn sprites, thus improving variety and appeal. A system that draws from the procedural benefits of speed towards map design, yet being a match with human artists in terms of finishing, would be a desirable one for content creators.

Generative Adversarial Networks (GANs) have been successful in various domains related to image and video including, for example, image generation (fashion items [6], portraits [7], and so on), paired [8] [9] and unpaired [10] image-to-image translation, inpainting [11] [12], super-resolution [13], image extension [14], transient attribute manipulation [15], video generation [16] and more.

The success of GANs led to the development of several

variant architectures. Of most benefit in this scenario are conditional Generative Adversarial Networks [17], which can be used to perform paired image-to-image translation between town designs and the final textured map. While describing a method for doing so, this paper will – moreover – present a novel post-processing algorithm to combine contiguous output towards increasing the final resolution, thus allowing high-resolution output to be generated from smaller, easily trained networks.

In line with the motivation stated earlier in this section, such a system could be paired with procedural techniques to increase the visual quality of their output, as shown in Fig. 1, by deriving town layouts from procedural generators as well as from user input. Early experiments on designing new town layouts also using AI have been carried out but this is beyond the scope of the current paper.

The rest of this paper is organised as follows. Section II briefly covers the Generative Adversarial Network paradigm and compares with some similar work. Section III covers dataset preparation, the training process and the post-processing algorithm which is central to this research. Section IV presents a sample of output from the algorithm, along with subjective and segmentation analyses. Section V discusses the impact of the algorithm parameters on the output and comments on the maximum resolution achievable by the algorithm.

II. Literature Review

A. Generative Adversarial Networks

A Generative Adversarial Network (GAN) is a paradigm for an unsupervised neural model that was introduced in 2014 by Goodfellow *et al.* [18]. GANs consist of two components: a Generator f(G), whose role is to learn the underlying distribution of a dataset, and a Discriminator f(D), whose role is to compare real samples from the dataset with fake examples from the Generator's output and attempt to distinguish between them.

The two components are engaged in an adversarial game where the Generator seeks to imitate the dataset samples well enough that the Discriminator cannot tell apart the difference. The Discriminator's success in identifying generated samples provides positive feedback for the Discriminator itself, and negative feedback for the Generator, forcing the latter to learn new patterns authentic to the dataset that the Discriminator cannot yet distinguish. This encourages the combined system to continuously improve itself until it reaches an equilibrium. In an optimal setting, the Generator and Discriminator must be balanced.

The original input to such a system is a random noise vector that determines the variety generated in each run of the network. Later, [8], based on the work described in [17], introduced a GAN variant where the input to the network is an image – known as a semantic layout – that conditions the network on which of the supported classes should be represented in the respective part of the image.

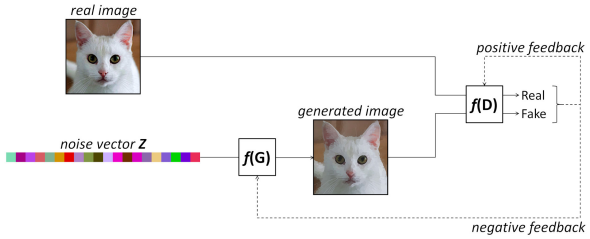

Fig. 2. A noise vector Z is passed through the Generator f(G) to produce an image that could potentially belong to the real dataset. Credit: Cat image taken from *Cats and Dogs* dataset [19].

These networks, known as conditional GANs (cGANs), allow the user to have more control over the generated output when compared to the original architecture. Randomness in cGANs is mostly obtained through the dropout of some nodes in the early upscaling layers.

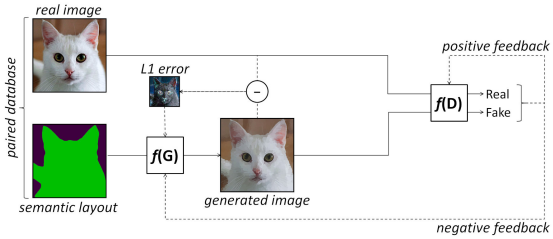

Fig. 3. A semantic layout specifies to the Generator which features compose the image, in this case: green is cat, purple is background. The L1 error (MAE) ensures that the system converges to the dataset colours as well as the high-frequency patterns. Credit: Cat image and semantic layout adapted from *Cats and Dogs* dataset [19].

B. Image-to-Image Translation

A major application of conditional GANs is in image-to-image translation, which is the task of converting an image from one domain to another. Examples of domains on which this technique has been applied include greyscale vs coloured images, sketches vs full colour output, real photo vs subject(s) with background removed, semantic labels vs full colour output, and so on. These examples are described in [8].

In order to train a cGAN to perform image-to-image translation, the dataset should consist of corresponding images taken in both domains. This is specifically known as *paired* image-to-image translation. (In contrast, *unpaired* image-to-image translation is an approach introduced in [10]; it does not require the images in either domain to be correlated, but generally performs slower.) After training is complete, the network would have learnt how to convert a previously unseen image in the source domain to what it would look like if it existed in the destination domain.

C. Similar Work

The concept of taking contiguous blocks of data as input and combining the results is not new but hasn't yet been experimented with in the area of generative networks. Some

similar, but not identical, concepts will be mentioned briefly here.

Stitching overlapping photos into a panorama view is perhaps most similar to the idea, but SIFT [20] and AI-based techniques (such as [21]) which are successful on real images are unlikely to work reliably in this scenario, since the stochastic nature of GANs – which is crucial to their generative potential – allows no guarantee that the same object will be rendered similarly enough in different outputs to be recognised as a feature marker. Along the same lines, one can argue about the similarity of this task to 3D scene reconstruction, as done in [22] and similar works. With regards to Super Mario Bros, [23] and related works used GANs to generate new levels, but in this case the contiguous blocks given to the network consisted of level data (not colour data) and were mostly self-contained, allowing the authors to simply attach the output without any processing. In fact, the cited paper does report that the only feature which is two units wide ('pipes') was visually inconsistent, possibly because it could span across different input windows. Finally, texture generation by GANs (such as [24]) is based on a similar concept and, being fully convolutional, does manage to remove architectural limits to resolution, but is designed to seamlessly preserve repeating patterns, which is not desirable when working with images.

III. Methodology

The context of 2d aerial fantasy town maps was chosen to test the proposed algorithm since maps are scaleable and contiguous. Some form of mapping context is therefore best suited for this algorithm. Nevertheless, it is expected that similar results to those demonstrated here can be achieved in other scenarios as well.

A. Features

Eight categories of features commonly occurring in aerial fantasy maps were selected as shown in Table I.

TABLE I
THE EIGHT FEATURES SUPPORTED BY THE TEXTURING CGAN.
PIX% STANDS FOR PIXEL-WISE PROPORTION OF THE FEATURES IN DATASET.

Category	Pix%	Examples
Water	13.8%	Rivers, Lakes, Sea, Waterborne vessels
Paths	17.9%	Walkable surfaces, ex: Dirt, Cobblestone, Bridges, Tiers
Grass	42.1%	Any type of grassland
Buildings	11.6%	Thatch, Wood, Stone buildings
Vegetation	5.3%	Trees, Bushes, Flowerbeds
Obstacles	0.9%	Man-made barriers, ex: Walls, Fences, Palisades
Terrain	3.8%	Sand, Soil, Farmland
Uninhabitable	4.6%	Cliffs, Rocks

Some features are more commonly represented than others, either because they naturally span large areas (such as grass and water) or because they are central to town mapping (such as paths and buildings). It is expected that the network will not perform as well for the less frequently represented features. For reference, the pixel-wise occurrence of each feature in the dataset is shown in Table I.

B. Dataset

The originality of the chosen context required the development of a new dataset. No other dataset is published for paired image-to-image translation which has both map layouts and textured output.

A small set of high-resolution maps were hand-labelled according to the features listed in Table I. The dataset used for training was derived by cutting overlapping 512×512-pixel windows from these labelled originals.

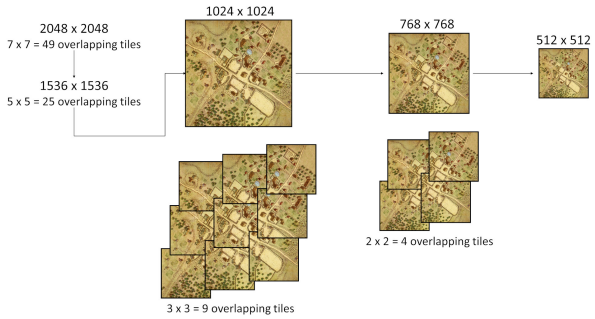

Fig. 4. Dataset augmentation from a few high-resolution maps. Each image is cut into contiguous and overlapping tiles. The image is scaled down and the process is repeated, exposing the network to some drastic changes in scale. For example, a single 2k×2k image can produce 88 tiles for the dataset. The same operations are done in parallel for the corresponding semantic layout. (Individual tiles for the higher resolutions are not shown in the diagram for the sake of neatness.) Featured map: [33].

For example, a 2k×2k image could generate 7 overlapping tiles of this size per dimension, four of which are contiguous and three are at the intersections.

In addition, the same process was applied after the originals were scaled to four smaller sizes, as shown in Fig. 4, to provide the networks with resilience against large changes in scale of features when drawing towns of different sizes.

The above preprocessing created a total of 2500 tiles, split in a 80:20 ratio between training set (2000) and test set (500).

An example of the input format, consisting of the cut image with the corresponding segmentation layout, is illustrated in Fig. 5.

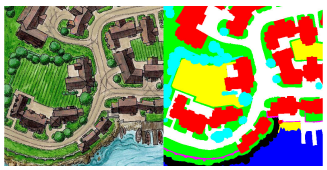

Fig. 5. The input to the system during training consists of the real tile next to the corresponding semantic layout. The Discriminator compares the Generator's output on the semantic layouts with the ground truths of the batch. Featured map: [34].

C. Architecture

The architecture utilised for this work was based on Tensorflow's implementation [25] of the Pix2Pix algorithm described in the original paper [8].

This implementation was intended for use in generating building facades from semantic layouts, one of the example datasets demonstrated in the paper. The authors of the paper itself demonstrated more applications of their algorithm, already mentioned in II-B. This implementation is adequate for the purposes of this work given that it deals with translating between semantic layouts of towns and fully textured maps.

The generator is a U-Net that consists of one added layer from the original implementation to increase the native architecture output to 512×512 pixels. All downsampling and upsampling layers consist of a filter with size 5 and stride 2. Batchnorm is applied to the first downsampling layer. Dropout on the first three upsampling layers serve to introduce randomness in the output and substitutes the noise vector for traditional GANs in this purpose.

The discriminator consists of five downsampling convolutional layers with stride 2 and two convolutional layers with stride 1. This discriminator has an array of 8×8 neurons with a receptive field of 381×381 pixels that output a score for how realistic their field is with respect to the dataset distribution.

The exact architectures are illustrated in Fig. 6.

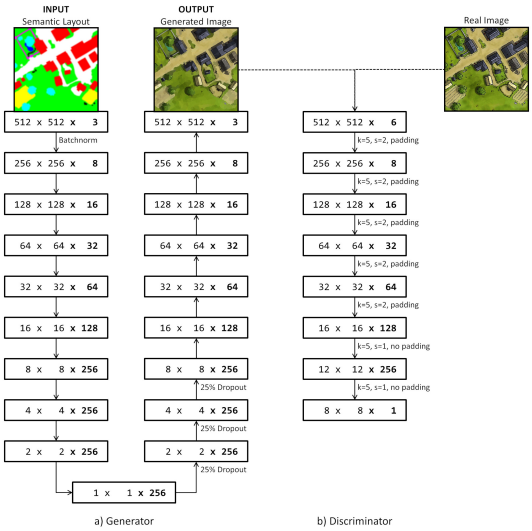

Fig. 6. The architecture used for this work is adapted from the Tensorflow implementation [25]. Original map: [35].

D. Training

Images in the dataset were further augmented live during training with jittering, rotation and flipping as per the original implementation.

All processing mentioned in this paper was carried out on a Windows device having an octacore processor, an Nvidia GTX1070 graphics card with 6GB onboard memory, and 32GB of RAM. The algorithm runs on the Tensorflow deep learning framework and uses Numpy, Numba and OpenCV libraries for fast array processing, operation compiling, and image manipulation respectively.

The GAN was trained for a total of approximately 6.5 hours (550 epochs at 43 seconds per epoch).

E. Blending

The architecture described above takes as input and generates as output images at a resolution of 512×512 pixels. The primary aim of this research is to allow the same network to draw larger images at a comparable quality that what one would expect from an image generated at the native resolution.

A possible first step in achieving this is to run the high-resolution input image as 512×512 tiles run separately through the network. Since the network does not have any information about the position of the tiles with respect to each other, it is not able to match colours at the edges properly. For this reason, the reconstituted image inevitably exhibits evident seams between the various tiles, as shown in Fig. 7 (third row of images).

The proposed approach involves running the original input multiple times through the network, each time dividing it into a different number of tiles per dimension. The underlying idea is that, if the seams occur at different places, one can compose the colour of each pixel in the final result from whichever runs do not have any seams close to that area. It is easy to ensure that the seams do not occur in the same areas of an image by tiling the input into a next larger prime number for each run, as shown in Fig. 7 (first row of images).

The main parameters of the blending algorithm are the number of runs through the network and the starting number of divisions. These parameters will be hereby referred to as *number of layers* (NoL) and *base tiling factor* (BTF) respectively.

For example, with NoL=3 and BTF=2, an input image will be submitted three times through the network, scaled and split into 2×2, 3×3 and 5×5 tiles. In this case, a total of 38 runs through the network are required to generate the higher resolution output.

The various layers are scaled back to the target resolution, and a fitness value is given to each pixel in each layer based on how close it is to the centre of its tile, measured as Euclidian distance, as shown in (1).

$$\phi_{(x,y,z)} = \sqrt{(P_x - \chi_x)^2 + (P_y - \chi_y)^2} \tag{1}$$

where χ_x and χ_y are the coordinates for the centre of the respective tile as determined with (2), and z is the layer index.

$$\chi_{(x,y,z)} = (\lfloor \frac{P_x}{T_W} \rfloor + 0.5) \times T_W \quad , \quad (\lfloor \frac{P_y}{T_H} \rfloor + 0.5) \times T_H \tag{2}$$

where P_x is the pixel's x-coordinate, P_y is the pixel's y-coordinate, T_W is the tile width and T_H is the tile height. An offset $(+0.5)$ is applied to the tile that the pixel belongs to $(\lfloor \frac{P_{x/y}}{T_{W/H}} \rfloor)$ so that it marks the centre.

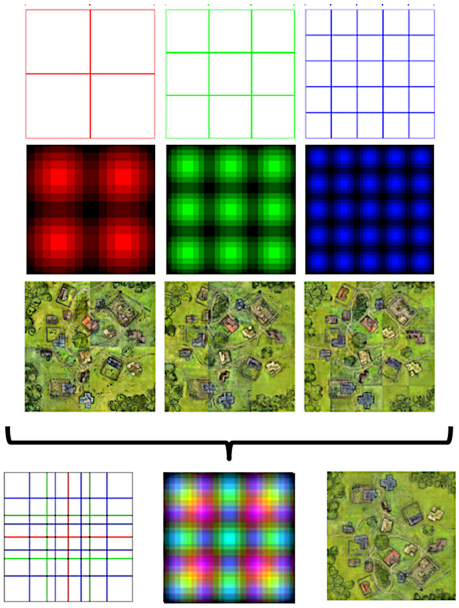

Fig. 7. This diagram demonstrates how the blending algorithm works. The same image is rendered multiple times using a different amount of tiles, in this example, 2×2 (left/red), 3×3 (centre/green) and 5×5 (right/blue). The seams created by tiling are removed by blending the various layers (bottom right). The fitness map in the bottom centre shows how the various layers are contributing to the final image. The semantic layout that produced this image was designed manually with the tool made for this research [26].

This fitness value is then normalised across all layers so that the lowest value is equivalent to 0 and the sum of the values in remaining layers amounts to 1, as shown in (3). In this manner, the layer with the lowest fitness (closest to an edge) does not contribute to the final image, improving the quality of the blending but requiring an extra processing layer (a minimum of 3).

$$\omega_{(x,y,z)} = \frac{\phi_{(x,y,z)} - \min\limits_{\phi(x,y,0\ldots n-1)}}{\sum_{z=0}^{n-1}(\phi(x,y,z) - \min\limits_{\phi(x,y,0\ldots n-1)})} \tag{3}$$

where $\phi_{x,y,z}$ is the fitness of the pixel at column x, row y and layer z, determined according to (1) and n is the number of layers (NoL) specified as a parameter.

The fitness map for the example given here is shown in Fig. 7 for the three layers separately (second row of images) and for the layers combined (bottom centre image).

Finally, for each pixel, the colour in the resulting image is given by the sum of the colours in all layers after they are multiplied by the respective fitness value, as shown in (4).

$$RGB_{(x,y)} = \sum_{z=0}^{n-1}(\omega_{x,y,z} \times RGB_{(x,y,z)}) \tag{4}$$

where $RGB_{(x,y,z)}$ is the colour of a pixel at a specific position in a particular layer z and $\omega_{x,y,z}$ is its contribution towards the colour of the same pixel in the final image, determined according to (3).

The result of this algorithm for the example in question is shown in Fig. 7 (bottom right image).

IV. Results

A. Examples

Fig. 8 shows the output of the algorithm on one of the maps used in the dataset.

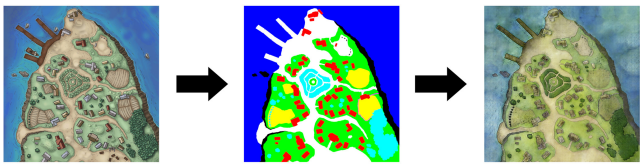

Fig. 8. One of the maps included in the dataset is shown on the left. The manually labelled semantic layout is shown at the centre. The algorithm's output on the semantic layout is shown on the right. Original map: [36].

The proposed algorithm was run on several samples from a number of unseen sources of map designs, including ones manually labelled through the tool specifically designed for this work [26], real-world town layouts from Google Maps [27], and a number of third-party procedural plain-colour town generators [1] [5] [28]. An example from each of these categories is shown in Fig. 9.

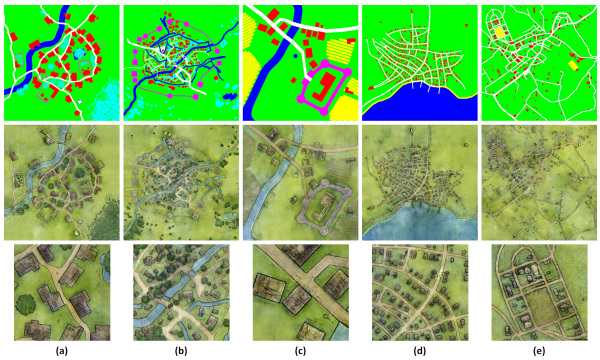

Fig. 9. Output from the network on (a) a manually designed map using [26], procedural map designs generated with (b) [1], (c) [28] and (d) [5], and a map derived from Google Maps [27]. The top row shows the semantic layout, the middle row shows the whole output, and the bottom row shows detail with a x4 zoom on the full image's centre.

B. Subjective analysis

45 participants were invited to a survey through groups dedicated to roleplaying games and cartography on social media, and all were shown the same set of images on the Google platform. For each image, the participants had to judge the perceived appeal of the shown map.

Four samples from this algorithm were included among a small but representative set of maps of different qualities and from diverse sources (procedural, made with mapping software and hand-drawn). The reason for a small overall sample was to maintain participant retention, while the choice to include only four samples from this algorithm was made to avoid

bias resulting from participants starting to recognise the maps generated by the GAN (equivalent to recognising style from the same artist). These four samples were selected arbitrarily and not on any criterion of quality; two were generated from manual designs, while the other two were generated from procedural layouts.

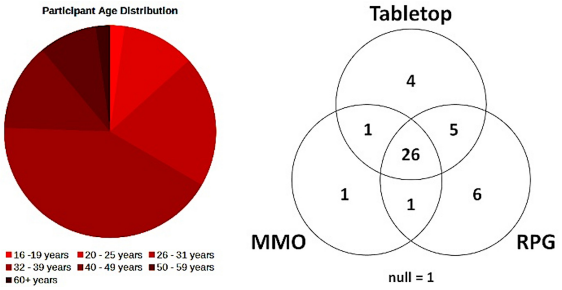

Fig. 10. The demographics of the survey population.

The vast majority of participants were 20-40 years old and had previous experience in at least some kind of roleplaying games (see Fig. 10). The average appeal score given by respondents was 3.49 on a Likert scale from 1 to 5. The most frequent score given by participants was 4 (Fig. 11, top).

After the source of the images was revealed, participants were shown the same layout rendered with different algorithm parameters to assess how these affect their perception of appeal in the resulting map. Results (see Fig. 11, bottom left) show that human observers prefer a low value for the number of layers (as illustrated by the spike on the '3' axis). However, one should point out that a lot of participants couldn't perceive a difference, or any differences didn't particularly add or detract from the appeal (similar spike on the 'N/A' axis).

On the other hand, participants significantly seem to prefer images processed with a base tiling factor of 3 (see Fig. 11, bottom right). This value has to be interpreted against the dimension of the images processed for the survey, which was 2048×2048 pixels. Higher resolutions would understandably require higher BTF values.

C. Segmentation analysis

Besides through subjective analysis, the output of the algorithm was also analysed by means of a separate U-Net trained on the dataset samples. Depending on how close the segmentation result is to the original layout, one can conclude to what extent the combined generative model and blending algorithm were successful at reproducing the original appearance of the features in the dataset.

Segmentation analysis removes the issues related with using humans for analysis, namely subjectivity and previous experience. However, it should be noted that the segmentation network had difficulty in recognising some poorly represented classes in the dataset.

After training, the segmentation model was run on two categories of samples: images generated from the *seen* whole-image semantic layouts labelled from the original artist drawings before being tiled for the dataset (which the network was

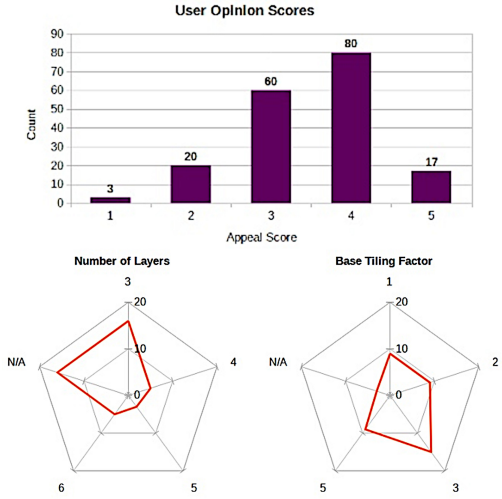

Fig. 11. Survey results: appeal scores (top), impact of NoL (left) and BTF (right) on the output.

exposed to albeit as tiles) and images generated from *unseen* semantic layouts from manual and procedural sources which were never presented to the network either in part or as a whole. Analysis on the seen samples (the former category) was included to provide a baseline estimate for the segmentation model's accuracy.

The confusion matrix on the segmentation results (Fig. 12) shows that the features which were most accurately represented by the system are environmental features which cover large areas, such as grass and water. Paths, houses and vegetation are also reproduced well by the proposed algorithm. These conclusions can be drawn from high values (darker colours) along the diagonals which show that the segmentation models correctly identified the feature for what it truly is. The system performs worst on the category of 'uninhabitable' (performance on unseen samples being significantly lower than on seen samples). On the other hand, segmentation analysis is inconclusive for the remaining two features (obstacles and terrain). In this case the segmentation model performed poorly on both seen and unseen categories and therefore this represents a failure in the segmentation model, not necessarily in the proposed technique.

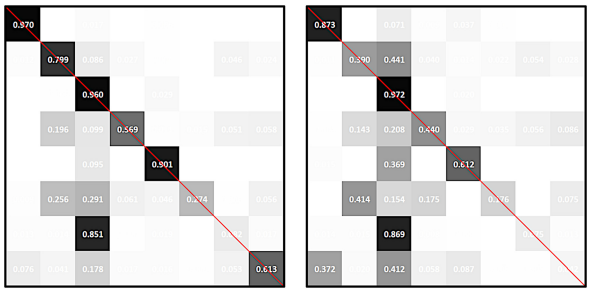

Fig. 12. Confusion matrix derived from segmentation results on seen (left) and unseen (right) images.

The average accuracy of all classes on the unseen samples is 0.448, if these are given the same importance toward this measure. If weighted by the occurrence of each class in the dataset, according to Table I, to compensate for typical class contribution, the accuracy of the texturing network is 0.689. Note that these calculations are made on the raw accuracies on the unseen category shown in small text in Fig. 12, without considering segmentation performance on the seen category.

V. Experiments

A. Resolution Limitations

The main aim of this research is to remove architectural limitations on the output of a Generative Adversarial Network. For this reason, to check whether the proposed algorithm is successful, an 8k×8k image (sixteen times the network size in either dimension) was run through it. The image was processed with 4 layers and a base tiling factor of 17 (the prime number closest to the magnitude of the image compared to network resolution).

With these settings, the inference stage requires a total of $17^2 + 19^2 + 23^2 + 29^2 = 2,020$ runs through the network and completes in around 40 seconds. The blending stage requires roughly 1 minute 15 seconds to blend the four layers with 67.1 million pixels each. It is important to note that, due to technical restrictions, the blending stage runs on CPU only (although the Numba package compiles the algorithm which somewhat improves performance), whereas Tensorflow allows graphics card acceleration for the inference stage.

The output is shown at various zoom levels in Fig. 13. This image is the same one featured at the beginning of this paper in Fig. 1.

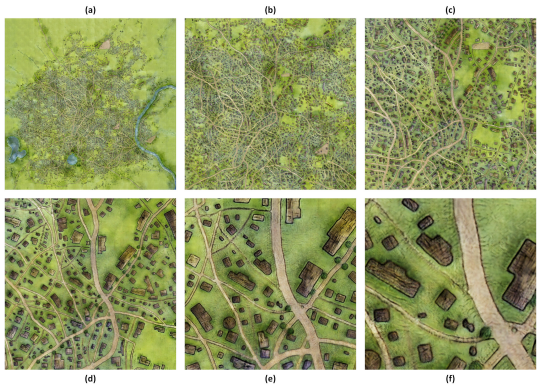

Fig. 13. An 8k image processed with the algorithm described in this paper. Each image shows a x2 zoom on the centre of the previous, starting from the full 8k×8k image in (a) to a 256×256 window in (f). The images show that quality is maintained even when zoomed in to this extent. The segmentation map used to produce this image was generated using [1].

B. Algorithm parameters

The algorithm's two main parameters impact the quality and characteristics of the output, as shown in Fig. 14.

Typically, a low number of layers (3 is the minimum) is characterised by sharper textures but may exhibit evident colour changes that reveal where the tiles have been joined together. A higher number of layers removes these artefacts but smooths out the detail as a result of more layers to blend together. The survey population seemed to prefer a low number of layers.

Fig. 14. This image shows how the output changes with different parameter values. The semantic layout that produced this image was designed manually with the tool made for this research [26].

Low base tiling factors result in coarser detail while higher values result in finer or subtle detail. Both extremes detract from appeal, as confirmed by the survey population. Generally, a base tiling factor equal to or around the ratio of the input image size with respect to the network size works best, as was done above (BTF around 16 for an 8k×8k image processed with an 512×512 network).

Both parameters lead to longer processing times if increased. The inference stage is affected by both NoL and BTF. The blending stage is affected by the NoL and the target resolution, although not by the BTF. The effect on parameters on processing times is shown below:

TABLE II
Times to process a 1024×1024 image with varying parameters.

Inference	**BTF1**	**BTF2**	**BTF4**	Blend	**BTF1**	**BTF2**	**BTF4**
NoL3	0.6s	1.1s	2.2s	**NoL3**	1.0s	1.0s	1.0s
NoL5	2.0s	4.1s	6.8s	**NoL5**	1.4s	1.4s	1.4s

VI. Conclusion and Future Work

This paper has presented a technique for using small, easily trained generative adversarial networks to create an image at much higher resolution than the native architecture can support by blending contiguous runs. The system parameters give the user control over smoothness of the blending and sharpness of detail. It was demonstrated that the technique manages to scale well, even to exaggerated resolutions such as 8k, provided that the input image has enough semantic detail for the desired size (that is, no large areas consist exclusively of the same feature). If this is allowed to happen, the system tends to retain seams, exhibit repeating patterns, or produce other artefacts detrimental to quality.

Given suitable input, however, reactions from human evaluators show that the texturing generated by the network is of sufficient quality to make it an improvement over town layouts created fully procedurally. This system can therefore be easily integrated with procedural algorithms to increase the appeal of the output while keeping the system fully autonomous.

One should point out that the current model suffers on texturing some categories which are poorly represented in the novel dataset. Future work in this regard would be a more comprehensive dataset that includes more examples from classes that either occur rarely or consist of fine details that are overwhelmed (space-wise) by more expansive features. The augmentation by tiling employed in this work helped avoid transformation issues, but not in providing sufficient distinct examples for training. Apart from this, one could also improve the visual appeal by incorporating saliency ranking [29] into the algorithm to identify more relevant features and process them differently than the rest of the image, such as by using a higher base tiling factor for increased detail on features that demand it.

In general, the base algorithm could be upgraded to incorporate recent advancements in GAN research, such as better normalisation techniques [30], pyramidal generator architectures to train the network in progressively larger sizes [7] [9] and differentiable augmentations [31]. Finally, a tile-blending technique based on inpainting methods may be attempted to check whether this improves performance and quality over the algorithmic method presented in this paper. Some experiments along this line for natural images are presented in [32].

Interested readers are invited to interact with the system presented here by means of the tool on [26].

REFERENCES

[1] Drachenzahn, "Roleplaying City Map Generator". Original link no longer available.

[2] Z. O. Toups, N. Lalone, S. A. Alharthi, H. N. Sharma, and A. M. Webb, "Making maps available for play: Analyzing the design of game cartography interfaces" in ACM Transactions on Computer-Human Interaction (TOCHI), vol. 26, no. 5, pp. 1–43, 2019.

[3] "Inkarnate". Retrieved on 09.03.2021 from https://inkarnate.com/.

[4] "Cityographer". Retrieved on 09.03.2021 from https://cityographer.com/.

[5] O. Dolya (Watabou), "Village generator". Retrieved on 09.03.2021 from https://watabou.itch.io/village-generator/.

[6] N. Pandey and A. Savakis, "Poly-gan: Multi-conditioned gan for fashion synthesis", arXiv preprint arXiv:1909.02165, 2019.

[7] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan", arXiv preprint arXiv:1912.04958, 2019.

[8] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks" in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134, 2017.

[9] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans" in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8798–8807, 2018.

[10] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks" in Proceedings of the IEEE international conference on computer vision, pp. 2223–2232, 2017.

[11] W. Xiong, J. Yu, Z. Lin, J. Yang, X. Lu, C. Barnes, and J. Luo, "Foreground-aware image inpainting" in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5840–5848, 2019.

[12] D. Seychell, C.J. Debono, "An Approach for Objective Quality Assessment of Image Inpainting Results" in 2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON), pp. 226-231, 2020.

[13] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network" in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1874–1883, 2016.

[14] P. Teterwak, A. Sarna, D. Krishnan, A. Maschinot, D. Belanger, C. Liu, and W. T. Freeman, "Boundless: Generative adversarial networks for image extension" in Proceedings of the IEEE International Conference on Computer Vision, pp. 10521–10530, 2019.

[15] L. Karacan, Z. Akata, A. Erdem, and E. Erdem, "Manipulating attributes of natural scenes via hallucination" in ACM Transactions on Graphics (TOG), vol. 39, pp. 1–7, ACM, 2019.

[16] Y. Wang, P. Bilinski, F. Bremond, and A. Dantcheva, "Imaginator: Conditional spatiotemporal gan for video generation" in The IEEE Winter Conference on Applications of Computer Vision, pp. 1160–1169, 2020.

[17] M. Mirza and S. Osindero, "Conditional generative adversarial nets", arXiv preprint arXiv:1411.1784, 2014.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets" in Advances in neural information processing systems, pp. 2672–2680, 2014.

[19] Microsoft/Kaggle, Cats vs Dogs Dataset. Retrieved on 09.03.2021 from https://www.kaggle.com/c/dogs-vs-cats

[20] D.G. Lowe, "Object recognition from local scale-invariant features" in Proceedings of the seventh IEEE international conference on computer vision, vol. 2, pp. 1150-1157, 1999.

[21] J.S. Sumantri, I.K. Park, "360 panorama synthesis from a sparse set of images with unknown field of view", arXiv preprint arXiv:1904.03326, 2019.

[22] C. Yu, Y. Wang, "3D-Scene-Gan: three dimensional scene reconstruction with generative adversarial networks" in International Conference on Learning Representations (ICLR), 2018.

[23] V. Volz, J. Schrum, J. Liu, S.M. Lucas, A. Smith, S. Risi, "Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network" in GECCO '18: Genetic and Evolutionary Computation Conference (Kyoto, Japan), 2018.

[24] N. Jetchev, U. Bergmann, R. Vollgraf, "Texture synthesis with spatial generative adversarial networks", arXiv preprint arXiv:1611.08207, 2016.

[25] Tensorflow, "Pix2pix implementation". Retrieved on 11.08.2020 from https://www.tensorflow.org/tutorials/generative/pix2pix/.

[26] G. Siracusa, "Gantographer". Retrieved on 09.03.2021 from https://gstw20.eu.pythonanywhere.com/.

[27] Google, "Google Maps". Retrieved on 09.03.2021 from https://maps.google.com/.

[28] O. Dolya (Watabou), "Medieval fantasy city generator". Retrieved on 09.03.2021 from https://watabou.itch.io/medieval-fantasy-city-generator/.

[29] D. Seychell, C.J. Debono, "Ranking regions of visual saliency in rgb-d content" in 2018 IEEE International Conference on 3D Immersion (IC3D), pp. 1-8, 2018.

[30] T. Park, M.Y. Liu, T.C. Wang, J.Y. Zhu, "Semantic image synthesis with spatially-adaptive normalization" in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2337-2346, 2019.

[31] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, T. Aila, "Training generative adversarial networks with limited data", arXiv preprint arXiv:2006.06676, 2020.

[32] D. Seychell, C.J. Debono, M. Bugeja, J. Borg, M. Sacco, "COTS: A multipurpose rgb-d dataset for saliency and image manipulation applications" in IEEE Access 9, pp. 21481-21497, 2021.

Miscellaneous:

[33] Triboar village from Storm King's Thunder (Wizards of the Coast).

[34] Bourmout village (SirInkman).

[35] Biirumura village (Ashlerb).

[36] Pinepass village (David Barrentine).