
Proceedings Paper:
Spick, Ryan John, Demediuk, Simon Peter and Walker, James Alfred (Accepted: 2019) Naive Mesh-to-Mesh Coloured Model Generation using 3D GANs. In: Proceedings of Australasian Computer Science Week (ACSW'20). ACM. (In Press)

Naive Mesh-to-Mesh Coloured Model Generation using 3D GANs

Ryan J. Spick
rjs623@york.ac.uk
Department of Computer Science,
University of York
York, UK

Simon Demediuk
simon.demediuk@york.ac.uk
Department of Computer Science,
University of York
York, UK

James Alfred Walker
james.walker@york.ac.uk
Department of Computer Science,
University of York
York, UK

ABSTRACT

3D model creation forms a large part of the development process in 3D graphical environments such as games or simulations. If an unsupervised approach can be used to generate high-quality textured models the turnaround in these areas could be greatly improved. Advances in generative deep learning have been shown to understand even complex 3D structures, allowing neural networks to output generations learned from abundant model data. But there are no methods that incorporate colour channels into these techniques, an important factor when attempting to use the generations in an immersive environment. Proposed in this paper is an advancement on the initial voxel-based 3D generative adversarial network (GAN) learning to include colour within the output generated samples through adapting the channels of voxel inputs. Followed by the application of marching cubes to translate the voxel-based models into a naive coloured mesh. The method uses unsupervised learning but requires a target 3D textured model data set. The techniques shown in this paper were tested on a sparse collection of model inputs from a set of open access textured models. The method was tested on a data set of 24 variant models of fish. The outputs from the trained generative model in this paper show promising results, learning the shape and a variety of unique texture patterns.

CCS CONCEPTS

• **Human-centered computing** → *Systems and tools for interaction design*; • **Computing methodologies** → *Neural networks*; *Image manipulation*; Genetic algorithms.

KEYWORDS

Procedural Content Generation, Generative Adversarial Network, 3D Models, Deep Learning

ACM Reference Format:
Ryan J. Spick, Simon Demediuk, and James Alfred Walker. 2020. Naive Mesh-to-Mesh Coloured Model Generation using 3D GANs. In *Proceedings of Australasian Computer Science Week (ACSW'20).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

ACSW'20, Feburary 2019, Melbourne, VIC, AUS

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Model generation forms an interesting topic within computer vision, and particularly the games field, where virtual renderings are becoming more and more detailed, there is a growing need for quicker asset development. Previous techniques have aimed to answer this question through the use of collective tree based generation, where parts of prior models are disassembled and then reassembled with randomly picked model components to form "new" structures [13]. On top of this, there are pseudo-automated methods used frequently in filmography, with large scale multi-agent scenes sometimes being generated through PCG techniques. Films such as The Lord of the Rings, World War Z and I Robot have utilised an engine called MASSIVE. This engine uses state of the art PCG methods that are also used in games to render an array of unique background "actors". Though these techniques lack the ability to render a multitude of random object models and require user involvement to validate parameters.

Recently deep learning through generative techniques have lead to novel methods for the creation of objects from a near-endless domain. Such as 2D textures, art with transferred styles, photo-realistic faces and text-image generation [2, 6, 10, 16]. While there is little research in the field of 3D generative techniques, perhaps due to the inherently difficult and computationaly large data structures compared to the 2D counterparts.

In this paper we present a method of extending 3D Deep Convolutional Generative Adversarial Networks (DCGAN) by adapting the pre-processed data stage to a 4 channel voxel-based design. The data input allows the 3-dimensional DCGAN to learn both original voxel data format, and an extended colour representation for each voxel block. The implementation of this type of learning could allow art creators for various virtual environments, who have access to many hundred model assets currently, the ability to easily extend their database of model data which could be implemented into future games/virtual environments.

There is also an exploration of the use of sparse data sets, and how generative networks perform on creating new varied samples when only exposed to a relatively small number of inputs. The number of models used in this paper is 24 high-quality models from various artists. The process shows how the variant art styles between artists could be concatenated when inputting to a generative network, while still creating high-quality procedural outputs.

Voxel-based data structures allow for a far simpler augmentation with current generative networks, where there is a uniform data structure. Whereas mesh data files usually contain complex arrangements of data, which don't inherently work with more general network inputs.

Finally, the adaptation of commonly used algorithms for voxel-mesh translation known as Marching Cubes [7] will be shown to be applied to the voxel data. Extrapolating the face colours from the voxel grid across the mesh's face colours.

The structure of the paper is as follows; Background and related work are discussed in Section 2. The data set will be briefly outlined in Section 3. Followed in Section 4 will be an in-depth outline of the methodology including the network architecture and pre-processing of the data. Next results are summarised in Section 5, followed by a discussion and conclusion in Sections 6 and 7.

2 BACKGROUND

Asset generation in games can form a large bottleneck in development times. With improvements to the automation and supervision of engine based game creation, a large proportion of time is now spent creating assets for games rather than the underlying code. This section will outline previous work in regards to 3D model generation with the use of deep learning to improve automated content creation. Alongside a description of the various techniques needed to translate data into the format used in this paper.

2.1 Procedural Content Generation (PCG)

PCG or procedural content generation is a method of utilising techniques and algorithms to generate content through automated processes with a focus on randomness. With the definition explicitly excluding any content that has a manual creation process (using graphics engines, inbuilt editor tools) [12]. Though this is an arguable statement, as content generated with procedural techniques that are polished through manual involvement may still be classified as procedural. Creating a slightly unclear divide on what PCG encompasses.

2.2 3D Models

3D modelling is the creation of a representative structure that corresponds to a surface shape. The use cases for 3D models are largely exhibited within games, however, are also shown in simulations and computer-generated imagery for films. These models can be created through supervised or automated methods using specialist software.

The amount of time varies on artist experience and the quality of the model needed, but the consensus is that within games a vast amount of time is spent on the creation of new assets for upcoming games, even though companies may already have preexisting models, which are sometimes 'recycled' for future games, the large proportion are brand new assets created from scratch. With this in mind, the ability to automate the process by re-purposing current model files into ones that have not been seen before would save time for developers.

The task of automating 3D generation forms an entirely different problem to solve compared to the lower dimensional counterpart of 2D image generation. There have been shown various approaches using evolutionary and algorithmic-based techniques such as;

Using multiple sections of pre-existing models to reconstruct variant samples of the original data points [13], these appear to be variant but it's easy to notice over time if exact sections of the selected models are used frequently.

Another interesting approach is the translation from the commonly used mesh data into a format that is much simpler to represent, mesh files have a complex list of faces, vertices, textures and sometimes normals, into a 3D array of values known as a voxel [15]. A voxel is a data structure that usually contains a binary value for each position in a 3D grid. With such a simple structure complex shapes can be rendered with no prior knowledge to object file data structures, however, depending on the complexity of the grid used to represent the mesh, the converted voxel quality could be reduced. An example of a voxel data structure can be seen in Fig. 1.

With this representation technique more modern-day approaches have been applied to voxel data, with generative adversarial networks being shown to accurately map and reconstruct input voxel data. This can be seen in [15] and [14] where a 3D variant of a Deep convolutional GAN can learn the structure of the voxels, which are inherently simple for the network to understand with only one boolean variable per position in a uniform grid.

A technique of avoiding the voxel conversion type learning has been shown in [3], where an end-to-end style learning approach is applied. Using two differing frequency generators and averaging the outputs results in a smooth surface mesh.

Consequently, none of these methods produces output results with colours learned within the network structure, some show post-processed texturing but these require manual or selected processing. The closest work related to this in the game-industry use is the use of the WaveFunctionCollapse algorithm[4], which is often applied to 3D models that were intentionally authored at the level of voxel.

2.3 DCGAN

GANs provide a unique framework that utilises two deep neural networks: a generator (G) network which attempts to capture the distribution of the training data, mapping this on to an input of variant noise (latent space) and a discriminator (D) network that will estimate the probability of its input being from the original training data or from the generators "recreated/forged" output, basically a discrete multilayer perceptron classifier [1].

The initial work on GANs used a multilayered perceptron model as their generators and discriminators networks. A method proposed in 2016 [9] built on the work done by Ian Goodfellow et al. by replacing the MLP networks with convolutional neural networks (CNNs) [5] creating a new architecture known as Deep Convolutional GAN or DCGAN. This architecture allowed breakthroughs in the way images were generated with more consistent results, previously producing noisy and incomprehensible outputs. Using three differences over classical CNN's the GAN methodology was able to utilize the power of convolutions in a more stable environment; pooling layers were replaced with strided convolutions [11] (Strided convolutions can be useful for when spatial information is not relevant. Whereas traditional pooling allows the network to forget about spatial structure of inputs), batch normalization (Batch normalization normalizes inputs of a network to zero mean and unit variance.) is used on inputs of *both* networks, there are no fully connected layers in between input and output layers of the network.

3 DATA SET

3.1 Data Set

The data gathered in this paper consists of 24 high quality textured 3D models of fish from 8 different artists, with noticeably different design styles from the 3D model repository, Turbosquid. With the approach also aiming to investigate how different art styles could be combined for the training of a generative model.

The intuition behind using varied 3D model styles was the ability for 3D artists to collaborate and perhaps combine their various styled data sets together. With the network learning the distributions of a wide variety of inputs, the outputs of the network could be manipulated for a particular style of model. The data set was also unusually sparse with few sample data points for learning using generative networks.

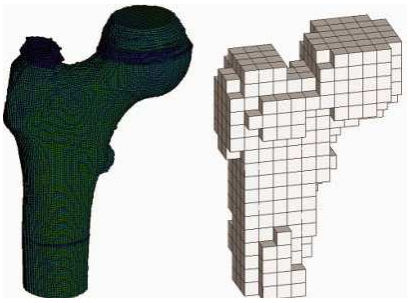


Figure 1: Example of a high quality mesh to a lower dimensional voxel structure [8].

4 METHODOLOGY

In this section, there will be an initial outline of the pre-processing method, and how the data set was manipulated to fit in with the type of network model proposed.

We adapt a DCGAN by changing the dimensions of the filter space from 2D to 3D by adding an additional depth channel. The importance of creating a simple input for the network is crucial to the correct learning and understanding of the data. Therefore the input to the network was decided to be in the format of a 3D tensor of colour values.

4.1 Pre-processing

For the model pre-processing the data was split from the usual format of voxel data, combining two approaches of using a standard RGB channel and a fourth channel to determine if a voxel appears in that space or not, this can be understood as a binary clamped alpha channel.

The models were all run through a voxelization process, by running a uniform 3D kernel over the mesh to calculate if a block should exist there or not. An example of a low-resolution conversion can be seen in Fig. 1.

Different to usual conversion methods the colour of the meshes face at the point of calculation was also carried over. This allowed the voxel to contain the additional information of the texture file. This conversion can be seen in Fig. 4b.

Table 1: List of the main DCGAN hyperparameters for the results shown in this paper.

Parameter	Value
Optimizer	ADAM
Learning rate (G)	0.005
Learning rate (D)	0.00005
Momentum	0.7
L2 regularization	$1e^{-5}$
Input Size	64^3
Batch size	8
Depth (N)	5

The voxel data was then 'hollowed' out to remove the data from the central point of the voxel data. The method was tested with both hollow and filled voxels, though there was no way to apply colour to the inside of the voxel while still not adding unnecessary noise to the network's inputs. It was possible to fill in the voxel with either one coloured blocks or an interpolation from the outside points, but for the purpose of mesh generation, the only required data was the voxel outer shell.

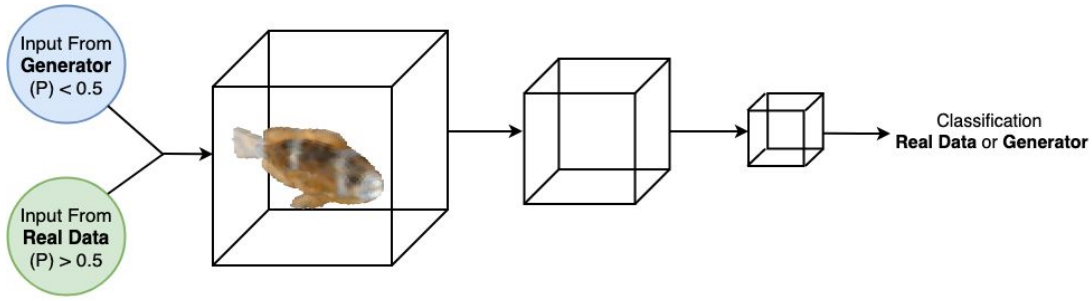
As the model was in 3 dimensions it was important to keep the size of the voxel relatively small such that the number of parameters didn't scale out of scope for the computer hard-wares capability. Usually, RGB generative inputs in 2d space have data size of $n^2 * 3$ With the proposed data structure in this paper the data size increases to $n^3 * 4$, therefore we chose to use size 64 in every dimension such that the visual fidelity of outputs was still high, but the size of the parameters could still fit inside of the hardware that would be used.

Though this was the size of the training input data, the actual size of the models varied and could be smaller. The $64x64x64$ grid simply acted a contained uniform box for all of the training data, each model was inserted into the larger sized grid around the central point creating a normalised size for each input.

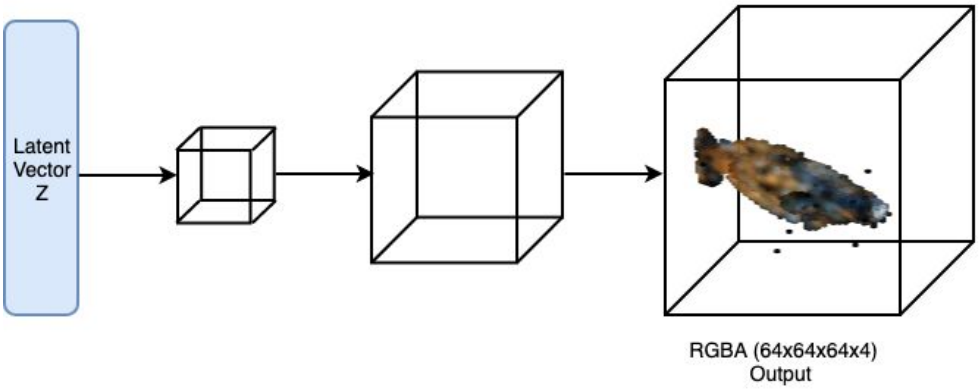
Finally, since the data had been inserted into a new 3D grid, it was important that the values were uniformly set. Such that the values that are not seen have a normally distributed random colour assigned. This data can be visualised as RGBA with the A channel being either -1 or 1 based on if a voxel appeared in that position. Accompanied by the RGB channel which contained the colour of the voxel scaled between -1 and 1 centered at 0 and if there was no voxel in that position then the RGB was set to a normally distributed random set of values between -1 and 1 to ensure the network was not saturated with one set of values, an example of a cross-section of a model training data can be seen in Fig. 3. Originally the non-appearing voxel values were set to a specific value of -1 (black) though with the sparsity of visible voxels the network could not learn the structure of the input data.

4.2 Network Training

The network used was a deep convolutional generative adversarial network (DCGAN) [9]. With the adaption of increasing the dimensionality by 1 channel, to 3 dimensions. The GAN consisted of two

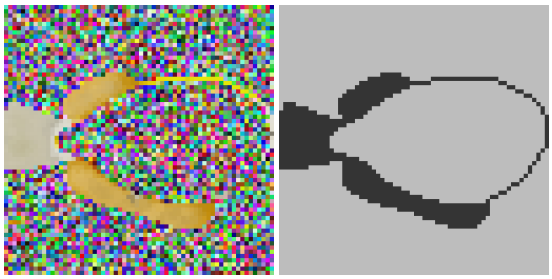


(a) Discriminator (Final output Sigmoid activation)



(b) Generator

Figure 2: Example of the network architectures for both the generator (b) and discriminator (a). Depth of 5 was used in both networks, however, a smaller representation is shown here. Each up and down sampling is twice or half the previous tensors size respectively.



(a) RGB Colour Channel **(b) Alpha Colour Channel**

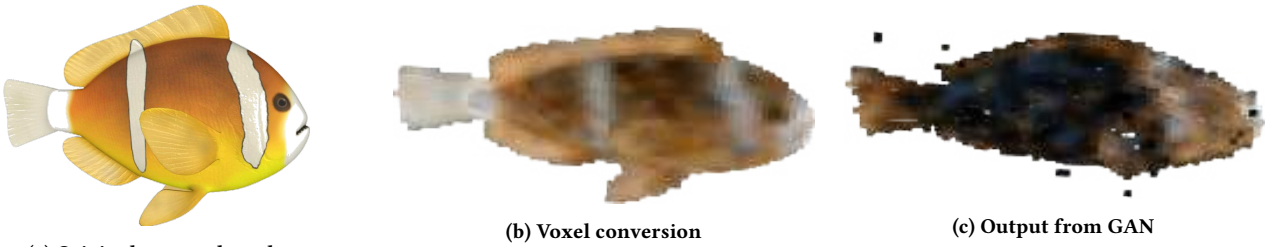
Figure 3: Central slice of an example of one of the training 3D voxels in the X-axis. With a separated visualisation of the data preparation of a 3D fish model. This shows the assignment of a random pixel colour for spaces that do not contain data to reduce the chance of sparsity of the learned region's colour.

networks; a generator which would create new samples from a latent vector of size 100 through a 5 depth up-sampling network with relu used between every hidden layer. A discriminator was used which would classify these samples, and a batch of the real data points, as either generated or real. The discriminator down-sampled from a 64^3 to a final fully connected sigmoid output. The exact network paramaters of both can be seen in Table 1 and an overview of the up-sampling / down-sampling of the two networks can be seen in Fig. 2.

There was a large degree of importance on parameter selection of the network. With only a sparse data set, using a higher learning rate for the generator, over the number of epochs the model was trained for, created a decidedly overfit model. With output generations hardly varying in visual appearance compared to the training data. While a large reduction to the generator's learning rate proved to be beneficial in generating more varied samples.

The network also employs a technique to reduce overpowering of the discriminator, such that when the discriminator's accuracy achieves over 75% the network halts the training of the discriminator.

The training of the network took 10,000 epochs to fully train the relatively small data set, taking 5 hours on one GTX 1080. The addition of the colour channels emphasised the time in which the network took to train, on top of decreasing the visual fidelity of the outputs compared to the original points.



(a) Original textured mesh

(b) Voxel conversion

(c) Output from GAN

Figure 4: Data pipeline from left to right. Showing the original textured mesh converted to a voxel which is used as an input to the network. Finall, an output from the fully trained network can be seen.

The network was understood to be fully trained when the generated outputs were observed to contain a similar amount of alpha points (110%) equal to 1 (signalling visible pixels) to an average of the original inputs. Though this was subjective and used as simply an indicator that the network was generating coherent outputs. The network saved checkpoints at 500 epoch intervals as to visually observe the outputs.

4.3 Marching Cubes Mesh Conversion

Marching cubes is an algorithm for constructing 3D surface models (meshes) from 3D voxel data [7].

Marching cube examples typically extend only the surface mesh, excluding any face colours. While if the face colour is stored alongside the surface mesh a naive extrapolation of the voxel's colours could be transferred into the mesh output. An example of this can be seen in Fig. 5.

Though the appearance of the marching cubes mesh is currently coarse, this could be alleviated through the use of hardware with larger memory capacity. This would allow inputs to exceed the input size of 64 used in this paper, which would produce a more smooth marching cubes conversion.

Figure 5: Low resolution mesh conversion using marching cubes with extrapolated face colours.

5 RESULTS

The results show a promising contribution to model generation with inherent learned texturing within the network. In Fig. 4 the pipeline of this technique can be seen, with the final image Fig. 4c showing a typical output of the network. With the network only able to learn from a sparse collection of inputs, the output models were remarkably varied in both colour and shape.

There was a varying amount of noise that existed around the output image which was an artifact of training. While applying a Gaussian blur in 3D space of kernel 3x3x3 and a low sigma of 0.05 pushed these artifact values below the alpha thresh-hold to show when rendered. A Gaussian blur was possible in this data due to the visible noise clusters existing in sparse areas away from the main model's distribution.

As mentioned previously the use of a marching cube algorithm provided a translation technique to transform the 3D voxel-based data into a naive low-quality mesh render. This type of mesh could be used within background renders of virtual scenes, where necessarily high detail is not needed, but the overall shape and colours can still be recognised from afar.

Previous results in this field share the same model structurally similarities [15], however, lack the importance of texturing/colouring the model. The results shown here visualise both structurally sound model outputs alongside an interesting application of the learned textures of the models. Though it is difficult to measure the "similarity" of the generations to the inputs, a random selection of the network generations can be seen in Fig. 6.

6 DISCUSSION

Current limitations with this technique revolve around the quality of the outputs, whether it be the voxel or the mesh conversion there is a distinct lack of size and quality of learned generations compared to the original.

With this in mind future work will be centred on applying super-resolution techniques that have been used within 2D face generation using GANs [6], to provide a computationally cheap upscaling to the voxel-based generations. With the better quality of voxel providing a more detailed mesh conversion when applying marching cubes.

Furthermore, many voxel games use direct block-based models, in which case the generative outputs could be used "out of the box" without any prior manipulation.

Figure 6: A random selection of output generations from the network.

7 CONCLUSION

Overall the results in this paper show an interesting contribution to 3D model generation with the inclusion of colour. Through a mesh to mesh style learning using a data translation technique to convert mesh data to voxel data, passed through a 3 dimensional, 4 channel generative adversarial network. Finally using marching cubes to convert from the voxel network output back to a naive coloured mesh of the learned data.

The novel contributions of the paper are the ability to generate 3D models with the added data channels to exhibit coloured texturing on 3D voxel outputs. With this extending to the application of marching cubes for textured mesh generation.

The exploration of using a spare data set with few sample points while still maintaining a high visual fidelity and generation variety proved successful. Utilising network parameters to control how quickly the generator could learn as to not over-fit on the few samples available to it.

There is an observation of noisy data within the visual standards of the output images, which indicates that the network struggled to understand the high dimensional data, especially with the additive colour channel compared to previous techniques.

ACKNOWLEDGMENTS

This work was supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Games Intelligence (IGGI) [EP/L015846/1] and the Digital Creativity Labs (digitalcreativity.ac.uk), jointly funded by EPSRC/AHRC/Innovate UK under grant no. EP/M023265/1.

REFERENCES

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[2] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. 2016. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207* (2016).

[3] Chiyu Jiang, Philip Marcus, et al. 2017. Hierarchical detail enhancing mesh-based shape generation with 3d generative adversarial network. *arXiv preprint arXiv:1709.07581* (2017).

[4] Isaac Karth and Adam M Smith. 2017. WaveFunctionCollapse is constraint solving in the wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM, 68.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[6] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4681–4690.

[7] William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, Vol. 21. ACM, 163–169.

[8] Phil Ong. 2013. Data Structures- Queues,Stacks,Voxels. http://philiponguoitgamedev.blogspot.com/2013/10/data-structures-queuesstacksvoxels.html. [Online; accessed 25-May-2019].

[9] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[10] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. 2016. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396* (2016).

[11] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).

[12] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N Yannakakis. 2011. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd international workshop on procedural content generation in games*. ACM, 3.

[13] Oliver Van Kaick, Kai Xu, Hao Zhang, Yanzhen Wang, Shuyang Sun, Ariel Shamir, and Daniel Cohen-Or. 2013. Co-hierarchical analysis of shape structures. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 69.

[14] Weiyue Wang, Qiangui Huang, Suya You, Chao Yang, and Ulrich Neumann. 2017. Shape inpainting using 3d generative adversarial network and recurrent convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2298–2306.

[15] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*. 82–90.

[16] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. 2223–2232.