# **Interactive CPPNs in GLSL**

Xavier Snelgrove\* Generative Poetics Toronto, ON, Canada hi@wxs.ca Matthew Tesfaldet<sup>†</sup> York University, Vector Institute Toronto, ON, Canada mtesfald@eecs.yorku.ca

## Abstract

Compositional Pattern Producing Networks (CPPNs) have the same input/output structure as OpenGL Shading Language (GLSL) fragment shaders, and can be easily implemented in GLSL for real-time interactive neural image generation. Because GLSL is standard, we find that this allows us to drop these CPPNs into a wide variety of popular tools for new media art, and use them as part of broader interactive real-time art pieces.

## 1 Introduction

The machine learning community has long looked at algorithms for image creation [1], with a recent resurgence of interest using new approaches such as feature visualization [2; 3], Generative Adversarial Networks (GANs) [4], and neural style transfer / texture synthesis [5; 6; 7].

If these new algorithms can be brought into the ecosystem of tools used by the graphics, new media arts, and games communities, then not only will these algorithms become more accessible to the large number of creators working in these ecosystems, but they will also be able to benefit from the other tools used in these communities.

Unity [8] is a leading real-time engine used by both the games community and the visual art community. OpenFrameworks [9] is very popular creative coding framework, as is Processing [10]. TouchDesigner [11] is used by Video DJs (VJs) and interactive installation designers. Three.js [12] is used to create online interactive 3D experiences. All of these tools contain support for the OpenGL Shading Language (GLSL) for GPU accelerated image creation and processing.

In this work, motivated by neural image synthesis applied in live performance, we build a translator that takes the weights of a trained Compositional Pattern Producing Network (CPPN)—a neural network architecture designed for visual art generation—and converts it into a GLSL fragment shader, allowing it to be easily "dropped in" to all of the systems described above to generate video in real-time either for direct display or as part of a broader display piece.

## 2 Review

#### 2.1 Compositional Pattern Producing Networks (CPPNs)

CPPNs show much promise for visual art generation [13; 14]. These are networks which map (x, y) pixel coordinates to (r, g, b) colour values via the composition and combination of various simple functions. Recently, there has been growing interest in using CPPNs as powerful differentiable image representations [15]. They allow for high resolution output as they can be sampled at arbitrarily fine spacings of (x, y), and have an inductive bias that we find visually pleasing — even randomly

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

<sup>\*</sup>http://wxs.ca

<sup>&</sup>lt;sup>†</sup>https://mtesfaldet.net/



Figure 1: Left: still from an audio-reactive CPPN running in ShaderToy in the browser (see https: //www.shadertoy.com/view/lttBz2). Middle: still from a CPPN mapped onto an object in TouchDesigner. Right: Photo of GLSL CPPNs projected in a live show.

selected parameters are interesting [16]. When used as the image generation component to visualize classification network features, they produce beautiful, more representational, work [15; 17].

## 2.2 Shaders

Pixel shaders (or fragment shaders) are programs that are used in the classic graphics pipeline to compute the colour of individual pixels on a surface. The program runs independently at every pixel on the surface and is passed context such as the pixel coordinate, the surface normal, and the lighting direction, which it can use to compute a colour.

GLSL is one popular language for writing these shaders, and allows them to be run efficiently on any GPU that supports OpenGL.

# 3 CPPNs in GLSL

CPPNs and fragment shaders are both functions which map (x, y) pixel coordinates to (r, g, b) colour values. This means CPPNs are well suited for implementation *as* fragment shaders [18]. In this work, we use the simple CPPN architecture from Mordvintsev *et al.* [15], which is a straightforward fully connected architecture. We provide code (https://github.com/wxs/cppn-to-glsl), building on Mordvintsev's TensorFlow [19] implementation for training CPPNs, and we convert it to GLSL.

#### 3.1 Motion and interactivity

For live performance, we need time-varying images. Although there are many advanced techniques for this, e.g., [20], we find that by perturbing the activations at early layers of the model, often the third or fourth, with a small offset, we get a pleasing continuous modulation of the generated image. In some work, we perform these perturbations as continuous functions of time to create animated textures; as a function of mouse position for interactive toys; or as a function of audio input for audio reactive visuals.

#### 3.2 Compatibility with existing tools

To show the versatility of the GLSL implementation of a CPPN, we experiment with it in a number of different platforms. Please refer to Fig. 1 for examples of GLSL CPPNs running in TouchDesigner, ShaderToy, and in live projection.

# 4 Conclusion

As the machine learning community interacts with the visual art and performance communities, we have much to learn from each other. By making our tools and techniques intercompatible we can both build on each other's wealth of knowledge and practice. This work is a small step towards that goal, as we build the components we need to start using neural image synthesis in a live performance practice.

### References

- A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 327–340.
- [2] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," University of Montreal, vol. 1341, no. 3, p. 1, 2009.
- [3] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," Distill, vol. 2, no. 11, p. e7, 2017.
- [4] 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*, 2014, pp. 2672–2680.
- [5] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in Advances in Neural Information Processing Systems, 2015, pp. 262–270.
- [6] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images." in *International Conference on Machine Learning*, 2016, pp. 1349–1357.
- [7] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European Conference on Computer Vision*. Springer, 2016, pp. 694–711.
- [8] "Unity." [Online]. Available: https://unity3d.com
- [9] Z. Lieberman, T. Watson, and A. Castro, "Openframeworks," 2009. [Online]. Available: http://openframeworks.cc/about
- [10] C. Reas and B. Fry, Processing: a programming handbook for visual designers and artists. Mit Press, 2007, no. 6812.
- [11] "Derivative TouchDesigner." [Online]. Available: https://www.derivative.ca/
- [12] R. Cabello, "Three.js," 2010. [Online]. Available: https://github.com/mrdoob/three.js
- [13] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, pp. 131–162, 2007.
- [14] K. Sims, "Artificial evolution for computer graphics," in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '91. New York, NY, USA: ACM, 1991, pp. 319–328.
- [15] A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah, "Differentiable image parameterizations," *Distill*, vol. 3, no. 7, p. e12, 2018.
- [16] D. Ha, "Generating abstract patterns with tensorflow," Feb. 2016. [Online]. Available: http://blog.otoro.net/2016/03/25/generating-abstract-patterns-with-tensorflow/
- [17] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 427–436.
- [18] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.
- [19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf
- [20] M. Tesfaldet, M. A. Brubaker, and K. G. Derpanis, "Two-stream convolutional networks for dynamic texture synthesis," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.