# ToonToy: a node-based NPR solution for UnrealEngine4

Emanuele Salvucci
*Revolution Software Ltd.*
York, United Kingdom
emanueles@revolution.co.uk

***Abstract*—We introduce "ToonToy" as a solution to the current lack of flexible and programmable non-photorealistic rendering solutions in modern game engines. The present paper briefly analyses stylistic rendering in games and introduces the problem of creating an "NPR-engine" rather than a single algorithm tuned to produce a single style. Whilst ToonToy undergoes continuous development, we describe general principles and some of the algorithms currently employed along with their technical implementation. We finally present results from commercial games currently in development as well as technical demos.**

***Index Terms*—NPR, games, image processing, real-time, Canny, XDoG, UnrealEngine**

## I. Introduction

Creating photo-realistic visuals in recent years has required game engines to implement the latest rendering technologies and incorporate ongoing research efforts. Nowadays, photorealistic real-time rendering is provided off-the-shelf by modern game engines. Recent experiments in cross-media productions [1] demonstrate how real-time rendering with these game engines could be used in place of traditional off-line rendering in the near future. Conversely, the development of non-photorealistic-rendering features is generally left with third-parties, if available at all on the market. Interestingly, a study [2] claims that there is not a lack of interest in stylistic games from the consumers, rather, there appears to be a technological "default bias" towards photo-realistic rendering. Whilst asserting that it is presently easier to develop a photo-realistic game than a stylistic one would make little sense in many aspects, we may broadly agree with Jarvis [2] and add that "3D rendering" was conceived, and still strives, to achieve synthetic photo-realism. Many game developers instead base their distinctiveness also on stylized graphics and rendering, often achieved through additional manual work such as hand-crafted textures, careful lighting, stylized character modeling and custom shaders.

## II. Motivation

ToonToy ("TT" hereafter) was initially conceived to obtain stylized visuals for "Beyond a Steel Sky", the sequel to the 1994 game "Beneath a Steel Sky". The global approach was to avoid additional artist intervention on a per-asset basis and hence TT was developed as a fully post-process solution operating exclusively on the G-buffers [3] provided in modern deferred rendering [4] pipelines. Also, the presented work shares some general considerations and motivations with "MNPR" [5], where a framework for stylistic rendering is described to be generic enough to incorporate different styles.

## III. Concept and Pipeline

TT is a complex screen-space post-process shading network created entirely within UnrealEngine's ("UE" hereafter) built-in node-based Material Editor (Fig.1). The fundamental concept behind TT is to define a rendering pipeline that simulates digital 2D drawing workflows based on three main stages: inking, coloring and compositing. The inking stage comprises four line rendering algorithms: a depth-test algorithm, a normal threshold algorithm and real-time implementations of Canny [6] and XDoG [7]. The Canny implementation also provides the ability to use Sobel [8] and Prewitt [9] kernels. Within the inking stage, hatching, cross-hatching and a custom bent-hatching algorithm are also included. In addition to using "hatching textures" inspired by Praun et al. [10], TT also implements generated lines that provide the ability to rotate, scale and vary line density, according to scene lighting. The coloring stage includes a cutout[1] algorithm using 1D LUT[2] textures, a multi-sample, displaced cutout and a cross-hatched cutout. All cutout algorithms can affect color textures or scene lighting selectively. Finally, the compositing stage allows for previous stages to be composited and blended with full-scene buffers provided by UE. TT also provides separate parameters for environment, characters, sky, fluids and foliage by indexing the stencil buffer. Moreover, a global "depth gradient" is used throughout all stages of the pipeline in order to modulate parameters based on the actual distance from the camera.
Finally, it should be noted how UE's peculiar shader-development environment, allowed us to integrate several specialized algorithms as mere contributions to each stage in the pipeline, as each algorithm can be selectively included or excluded by means of "switches". When a feature is switched on or off, UE automatically re-compiles the whole network into a new HLSL shader so that only a given set of features are evaluated to create a specific style, rather than the entire set.

[1]The "cutout" term is used here to identify a cel-shading effect.
[2]Look-Up Table.

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

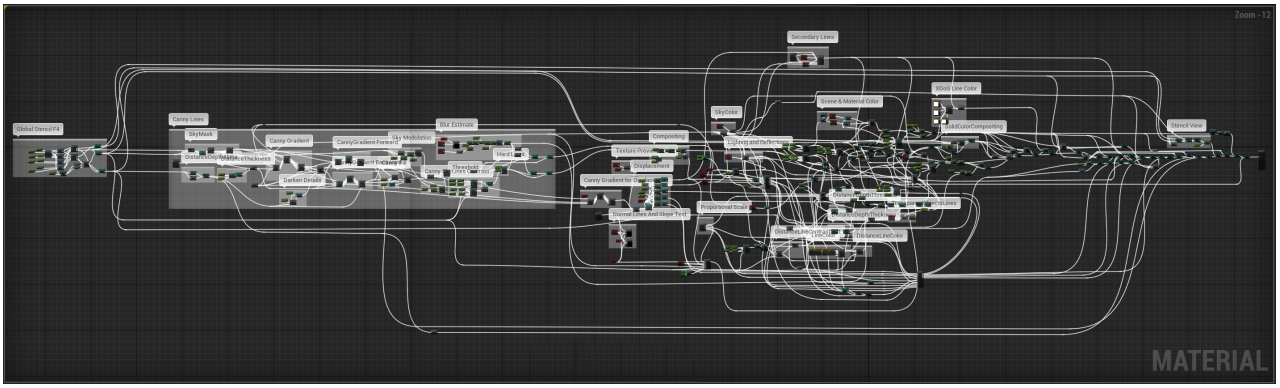


Fig. 1. ToonToy node network in UnrealEngine4. Many of the depicted nodes are functions containing hierarchically nested nodes.

## IV. Canny Implementation

The Canny edge detection algorithm has been used for decades in several fields of computer graphics, including NPR. Our implementation of the Canny algorithm was initially following the canonical real-time ones [11] [12], i.e. using multiple passes for non-maximal suppression although it was straightforward to implement it as a single pass and compare performance. In our single-pass implementation, forward and backward gradients are fully computed for each pixel instead of being looked up as in the multi-pass version. Table I shows

TABLE I
Performance of multi-pass and single-pass Canny at 1920x1080 on GeForce GTX 960

| | GPU Time (ms) |
|---|---|
| Multi-pass | 0.43 |
| Single-pass | 0.42 |

the two implementations in UE are substantially equivalent on modern entry-level GPUs in terms of performance.
Unlike 2D applications, parameters for the Canny operator must be adjusted to compensate for high variations in spatial density, and hence frequency, of the input. This is especially noticeable when using the world normals buffer as the input to the Canny operator, as texels are filtered and density varies according to depth. As shown in Fig.2, we use a depth gradient to drive and adjust Canny parameters in order to keep rendered details as homogeneous as possible in most cases.

## V. Artistic Considerations

As depicted in Fig.3, some settings of the Canny operator produce line rendering that is reminiscent of the work of renowned artist Jean Giraud, aka “Moebius”, where lines appear detailed and thin and play a major role in defining the overall look.

Fig.4 shows how the XDoG contribution instead may be associated with anglo-saxon comic book styles, with black and white lines, large inked areas and overall providing a harsh high contrast, reminiscent of the seminal work produced by renowned comic book artist Dave Gibbons.

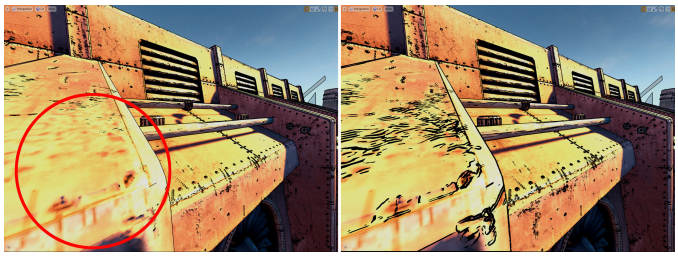

Fig. 2. (Left) Loss of details within the red circle due to magnified normal map texels. (Right) Canny parameters are offset-ed by the depth gradient to recover details.

Fig. 3. (Top) Original rendering. (Bottom) ToonToy style inspired by Moebius applied.

Fig.5 also shows how tuning of XDoG parameters can also produce very different results.

Fig. 4. (Top) Original rendering. (Bottom) ToonToy rendering employing XDoG as a contribution to the inking stage. Assets from "Infinty Blade" [13], courtesy of Epic Games.

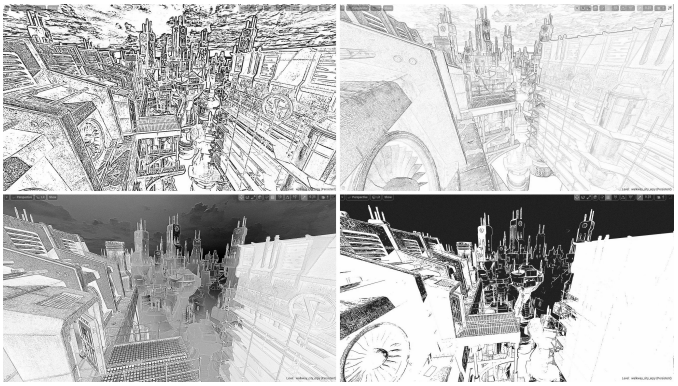

Fig. 5. XDoG early tests produced with ToonToy showing different rendering styles in the inking stage only.

## VI. Results

We present additional results from "Beyond a Steel Sky" (Fig.6,7), currently in production, and from the tech demo "Paragon Tribute" (Fig.8), a 150-seconds animation produced with TT and UE using "Paragon" assets [14] by Epic Games, demonstrating TT cross-media capabilities with any kind of game assets without additional preparation work.

## VII. Conclusions

TT presently comprises more than 500 parameters controlling all aspects of the rendering pipeline. It is HDR compliant and provides artists with an in-game interface to build-up their styles library, change styles on-the-fly and associate standard UE post-processing parameters with the current TT style.

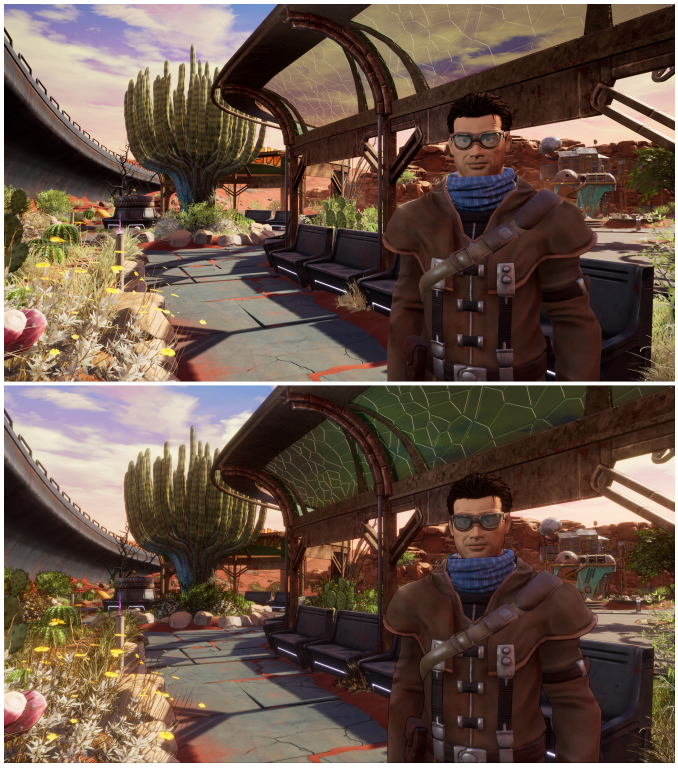

Fig. 6. Screenshot from "Beyond a Steel Sky". (Top) Standard rendering. (Bottom) ToonToy style applied.


Fig. 7. Screenshot from "Beyond a Steel Sky". (Top) ToonToy style applied. (Bottom) Inking stage only.

Fig. 8. Screenshots from "Paragon Tribute'. (Top) Original UnrealEngine4 rendering. (Others) Different ToonToy styles applied without additional external compositing. Assets from "Paragon" [14], courtesy of Epic Games.

As any content can be turned into a stylized rendering without additional manual work and in real-time, new cross-media productions can be explored, including production of cartoons and comic books straight from games assets using the game engine as a renderer.

## Acknowledgments

The author wishes to thank the following people and organizations:

- Charles Cecil: as the principal supporter of the present research.
- Sucha Singh: for creating the ToonToy style at the heart of "Beyond a Steel Sky" and art assets in Fig.2,4,5.
- Dave Gibbons: for precious suggestions and for being a source of inspiration.
- The whole team at Revolution Software.
- Epic Games for releasing such great content [13] [14] to the developers community for free.

## References

[1] "The human race," The Mill, Epic Games, 2017, accessed: 2019-02-09. [Online]. Available: http://www.themill.com/portfolio/3516/the-human-race-

[2] N. Jarvis, "Photorealism versus Non-Photorealism: Art styles in computer games and the default bias," Master's thesis, University of Huddersfield, 2013.

[3] T. Saito and T. Takahashi, "Comprehensible rendering of 3-d shapes," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90. New York, NY, USA: ACM, 1990, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/97879.97901

[4] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The triangle processor and normal vector shader: a vlsi system for high performance graphics," in *Acm siggraph computer graphics*, vol. 22, no. 4. ACM, 1988, pp. 21–30.

[5] S. E. Montesdeoca, H. S. Seah, A. Semmo, P. Bénard, R. Vergne, J. Thollot, and D. Benvenuti, "Mnpr: A framework for real-time expressive non-photorealistic rendering of 3d computer graphics," in *Expressive 18: The Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, ser. NPAR '18. New York, NY, USA: ACM, 2018, pp. 9:1–9:11.

[6] J. Canny, "A computational approach to edge detection," in *Readings in Computer Vision*. Elsevier, 1987, pp. 184–203.

[7] H. Winnemöller, "Xdog: advanced image stylization with extended difference-of-gaussians," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*. ACM, 2011, pp. 147–156.

[8] I. Sobel, "An isotropic 3× 3 image gradient operator," *Machine vision for three-dimensional scenes*, pp. 376–379, 1990.

[9] J. M. Prewitt, "Object enhancement and extraction," *Picture processing and Psychopictorics*, vol. 10, no. 1, pp. 15–19, 1970.

[10] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Real-time hatching," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, p. 581.

[11] J. Fung, "Computer vision on the gpu," *GPU Gems*, vol. 2, no. 649-666, p. 34, 2005.

[12] Y. Roodt, W. Visser, and W. A. Clarke, "Image processing on the gpu: Implementing the canny edge detection algorithm," 2007.

[13] "Infinity blade assets," Epic Games, 2015, accessed: 2019-02-09. [Online]. Available: https://www.unrealengine.com/marketplace/assets?q=infinity+blade

[14] "Paragon assets," Epic Games, 2018, accessed: 2019-05-10. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/assets?keywords=paragon