Digital Typography Rendering

Nicolas P. Rougier, Behdad Esfahbod

Digital Typography Rendering

25 years of text rendering

Nicolas P . Rougier
Inria
Talence, France
Nicolas.Rougier@inria.fr

Behdad Esfahbod
Google
Menlo Park, California, USA
behdad@behdad.org

Figure 1: 25 years of text rendering

ABSTRACT

This short course (1h30) is an introduction to digital typography rendering, providing key concepts of typography as well as introducing several computer graphics techniques to render text, from the oldest and most common techniques (texture based) to the latest methods taking full advantage of shaders with quasi flawless rendering.

CCS CONCEPTS

• **Computing methodologies** → *Rasterization*; *Non-photorealistic rendering*;

KEYWORDS

Text rendering, Bézier curves, shaders, typography, antialiasing, kerning, shaping, hinting, unicode

ACM Reference Format:
Nicolas P . Rougier and Behdad Esfahbod. 2018. Digital Typography Rendering: 25 years of text rendering. In *Proceedings of SIGGRAPH '18 Courses.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3214834.3214837

SIGGRAPH '18 Courses, August 12-16, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-5809-5/18/08.
https://doi.org/10.1145/3214834.3214837

1 INTRODUCTION

Typography is *the art of arranging type to make written language legible, readable, and appealing when displayed*[1]. However, for the neophyte, typography is mostly apprehended as the juxtaposition of characters displayed on the screen while for the expert, typography means typeface, scripts, unicode, glyphs, ascender, descender, tracking, hinting, kerning, shaping, weigth, slant, etc. Typography is actually much more than the mere rendering of glyphs and involves many different concepts. If glyph rendering is an important part of the rendering pipeline as it will be explainded below, it is nonetheless important to have a basic understanding of typography or there's a known risk at rendering garbage on screen, as it has been seen many times in games, software and operating systems.

2 TYPOGRAPHY

With the widespread adoption of Unicode[2] as the canonical character set for representing text a whole new domain has been opened up in the system software design. Traditionally fonts were a collection of glyphs and a simple one-to-one mapping between characters and glyphs. Rudimentary support for simple ligatures was available in some font formats. With Unicode however there was a need for formats allowing complex transformation of glyphs (substitution and positioning). Two technologies were developed to achieve that, one is OpenType Layout from Microsoft and Adobe, the other is AAT from Apple. These two technologies, plus TrueType and Type1 font formats, all were combined in what is called OpenType.

[1] According to Wikipedia
[2] http://unicode.org

There are fundamental differences in how AAT and OpenType Layout work. In AAT the font contains all the logic required to perform complex text shaping (the process of converting Unicode text to glyph indices and positions). Whereas in OpenType, the script-specific logic (say, Arabic cursive joining, etc) is part of the standard and implemented by the layout engine, with fonts providing only the font-specific data that the layout engine can use to perform complex shaping. The Free Software text stack is based on the OpenType Layout technology. HarfBuzz[3] is an implementation of the OpenType Layout engine (aka layout engine) and the script-specific logic (aka shaping engine).

Such complex shaping and layouting poses new problem in term of rendering. If it is possible to cache glyphs on the GPU for simple languages (e.g. English), it is virtually impossible to do the same with complex layouts where a tremendous amount of composed glyphs can be created on the fly. New approaches are thus necessary to ensure fast, correct and just-in-time rendering.

3 RENDERING

Near the end of the last century, [Kilgard 1997] introduced a simple OpenGL-based API for texture mapped text. The method packed many rasterized glyphs into a single alpha-only texture map and used a lookup table to assign texture coordinates to a quadrilateral to extract a glyph when rendering. This approach yielded several advantages over previous approaches (fixed size bitmap, stroke font) because it allowed arbitrary rotation, scaling and projection over 3D surface. More importantly, it allowed to have arbitrary shaped glyph that can be loaded directly from standard bitmap font files, such that font designers could work with standard tools instead of writing explicit code to render glyph shapes. The primary drawback of the method was that at high resolution, text became quite pixelated, even using bi-linear texture interpolation. To cope with this problem, [Green 2007] refined the method by computing a (non-adaptive) signed distance field in place of the rasterized glyph that could later be alpha-tested or thresholded. [Gustavson 2012] showed that distance field can be computed efficiently. The method of [Green 2007] differed from the one proposed by [Frisken et al. 2000] that was relying on an adaptive distance field and octree and did not take advantage of shader at the time of publication.

Unfortunately, at very high resolution, artifacts still appeared and it became necessary to use true vector font to achieve flawless rendering. Before the advent of programmable shaders, this was performed by approximating Bézier curves with many line segments, which was computationally expensive. However, [Loop and Blinn 2005] introduced a new approach for resolution-independent rendering of quadratic and cubic spline curves. By tessellating a glyph the proper way, they offered de facto a method for resolution independent rendering of a glyph with good rendering quality (anti-aliased is also supported). A few years later, [Esfahbod 2011] refined the signed distance field method by using arc approximation of Bézier curves and encoding the required information into a texture, allowing the conservation of sharp angles at any resolution. More recently, [Kilgard and Bolz 2012] proposed a two-steps method (stencil and cover) for GPU-accelerated path rendering as an OpenGL extension. As stated by the authors, their goals are completeness, correctness, quality, and performance, the extension actually covers most of the OpenVG specifications while giving very high performances. In the meantime, [Rougier 2013] proposed a method for 2d rendering achieving best possible result with sub-pixel rendering and positioning in the screen space. Two years ago, [Chlumsky 2015] made a breakthrough by proposing a multi-channel signed distance field method that was able to render sharp angles (smoothed out corners is quite characteristic of the regular signed distance field method). Last, but not least, two new methods have been introduced last year by [Walton 2017] and [Lengyel 2017] respectively. The former is currently tested for a future integration in the Quantum rendering engine used by Firefox while the latter has been integrated into the slug library. They both achieved fast and beautiful rendering and may have nearly solved the glyph rendering problem... Until a new technique appears.

[3]https://www.freedesktop.org/wiki/Software/HarfBuzz/

4 COURSE OVERVIEW

This 1h30 course has been split in two parts: one part introducing key concepts in digital typography as well as the related software stack (necessary libraries to read and use font information) and the second part about the various computer graphics techniques for rendering glyphs on screen.

- Part I: Introduction (25mn)
- Part II: Texture based rendering (20mn)
- Part II: Distance based rendering (20mn)
- Part II: Geometry based rendering (20mn)
- Conclusion (5mn)

REFERENCES

Viktor Chlumsky. 2015. *Shape Decomposition for Multi-channel Distance Fields.* Master's thesis. Czech Technical University in Prague,.

Behdad Esfahbod. 2011. Glyphy. https://github.com/behdad/glyphy. (2011).

Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. 2000. Adaptively sampled distance fields. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00.* ACM Press. https://doi.org/10.1145/344779.344899

Chris Green. 2007. Improved Alpha-Tested Magnification for Vector Textures and Special Effects. In *ACM SIGGRAPH 2007 courses (SIGGRAPH '07).* ACM, New York, NY, USA, 9–18.

Stefan Gustavson. 2012. *OpenGL Insights.* CRC Press, Chapter 2D Shape Rendering by Distance Fields, 173–182.

Mark Kilgard. 1997. A Simple OpenGL-based API for Texture Mapped Text, Silicon Graphics. (1997). http://reality.sgi.com/opengl/tips/TexFont/TexFont.html.

Mark Kilgard and Jeff Bolz. 2012. GPU-Accelerated Path Rendering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2012)* 31, 6 (Nov. 2012), to appear.

Eric Lengyel. 2017. GPU-Centered Font Rendering Directly from Glyph Outlines. *Journal of Computer Graphics Techniques (JCGT)* 6, 2 (14 June 2017), 31–47. http://jcgt.org/published/0006/02/02/

Charles Loop and Jim Blinn. 2005. Resolution Independent Curve Rendering Using Programmable Graphics Hardware. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05).* 1000–1009.

Nicolas P. Rougier. 2013. Higher Quality 2D Text Rendering. *Journal of Computer Graphics Techniques (JCGT)* 2, 1 (30 April 2013), 50–64. http://jcgt.org/published/0002/01/04/

Patrick Walton. 2017. Path Finder. http://pcwalton.github.io/blog/2017/02/14/pathfinder/. (2017).