Accelerating Film Environment Creation Using Game Development Tools

John Vanderbeck
MPC
john-van@moving-picture.com

Alex Jenyon
MPC
alex-je@moving-picture.com

Figure 1: Real-time layout and final shot comparison from Justice League ©2017 Warner Brothers. All rights reserved.

ABSTRACT

For *Justice League* MPC faced the challenge of creating an abandoned city nearly twelve square kilometers in size, and for *A Wrinkle in Time* a fast-moving forest sequence that required rapid iterations with closely managed art direction. To deliver the high quality that viewers expect with the control that film makers require we needed procedural, extensible tools - but still allow for detailed artistic control. In this paper we look at how MPC leveraged real time game engines such as Unreal Engine and Unity to produce massive environments, and speed up the iteration cycle of complex VFX.

CCS CONCEPTS

• **General and reference → Surveys and overviews**;

KEYWORDS

environment, world, layout, set dressing, real-time, unreal, unity

ACM Reference Format:
John Vanderbeck and Alex Jenyon. 2018. Accelerating Film Environment Creation Using Game Development Tools. In *Proceedings of SIGGRAPH '18 Talks*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3214745.3214775

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

ACM ISBN 978-1-4503-5820-0/18/08.
https://doi.org/10.1145/3214745.3214775

1 BRINGING A GAME ENGINE TO A PARADEMON FIGHT

Feature film environments are constantly growing - both in terms of size and complexity - and their creation in a VFX pipline presents several challenges. The best way to fill out large worlds is with procedural toolsets, yet a visual effects workflow must also allow an artist full creative control over all aspects of the shot. These two tenets seem at first to be at odds with each other, but with careful design it is possible to build procedural layout tools that still allow an artist to step in and manipulate individual entities as needed.

Commercial game engines such as Unity and Unreal have built-in game editors, purpose-built for large scale world-building tasks. Leveraging them for real-time set dressing and layout provides a great base to build upon, as the majority of the framework needed is already available, and the editor allows for the easy addition of custom tools or extensions.

The combination of a commercially available game engine, along with a set of custom pipeline and set dressing tools, proved to be a powerful workflow. The first project where this idea was implemented in full production conditions was *Justice league*, after early testing on *Independence Day: Resurgence*.

2 JUSTICE LEAGUE

In *Justice League*, a 12 square kilometer city required over 400,000 instances of set dressed props and two million trees to bring it to life. We saw this as the perfect opportunity to utilize our real-time set dressing toolset.

On this project, we used two primary tools inside the Unreal Engine editor named "Spline scatter" and "Foliage paint".

Spline scatter is one of a series of custom scatter tools built in "Blueprint", Unreal's visual scripting tool. It allows an artist to draw

a spline, and procedurally scatter large numbers of object instances underneath the spline path. Like most useful scatter tools, the objects are pulled from a weighted list, and have random rotation and jitter applied. Should the underlying geometry change, the position of the instances can be procedurally updated - but can also be manually edited by the artist if required. This system was built very quickly by leveraging visual scripting systems available in the engine editor - and could therefore be customized per shot if needed. An example of this fast tool iteration was an artist request for scattered instances (in this case the slabs of a sidewalk) to snap to an arbitrary grid. Adding this functionality and re-deploying the tool took only a few hours, and was in use on the show the same day.

While the scatter tools were useful, artists also painted instances directly into the world using the Foliage Painter system that was built into the editor. Both tools tended to be used in a complementary way - scatter systems for large, generic masses of instances, and foliage paint for more precise control.

Artist work in the editor was exported as layout information back to the pipeline for hand off to other departments, and rendered in lighting. An interesting challenge was the texturing of the terrain, which was painted in the editor using a 5-layer height blended shader. The blend information was exported as primvars to Renderman, and the shader was re-constructed in Katana to perfectly match the real-time renders.

3 PIPELINE INTEGRATION OF A GAME ENGINE

In a typical game workflow, the game engine is an end consumer of assets. Models, and textures are created in traditional ways, and imported into the editor. From there they are used in making a game level, and built directly into the game.

Using a game engine in a visual effects pipeline however means allowing it to work in the middle of the pipeline, both importing assets, and exporting them. No game is ever actually built, we simply use the editor as a powerful DCC. This was further complicated by the fact that MPC's pipeline is Linux based, whereas the game engines we were using are Windows based. Therefore, integrating these editors into the pipeline involved two main pieces; communication between the editor on Windows with the pipeline on Linux, and conversion of assets between in house formats and those used by the editors.

Cross platform communication was achieved through the use of ZeroMQ and a basic Request-Response setup. The server ran on Linux and did the majority of the work communicating with the pipeline, while a light weight client was built into the editors. Artists worked on Windows directly in the game engine and this tool, named MPCLink, worked behind the scenes to pull assets into the editor or push them back to the pipeline.

MPCLink lets the artist browse our in-house asset database, Tessa, from a custom native window inside Unity and Unreal. When an asset is chosen for import, MPCLink gathers all the necessary information from the pipeline, and converts our in-house formats to standard ones recognized by the game engine. As UDIM tiles are not supported in the game engine, they are automatically converted to multiple materials, and the asset looks and acts as if it was imported using the engine's standard tools. Layouts created in the game scene could then be exported back out to the pipeline through MPCLink. This process exports just the transform of each asset instance, along with a URI identifier that links it back to where it originated from, making it extremely fast. MPCLink then uses this information to build what MPC calls a Hierarchy Package, which is then sent to Katana for rendering.

4 DECREASING THE ITERATION CYCLE ON *A WRINKLE IN TIME*

A major benefit gained from set dressing inside of a game engine is that you get a powerful real-time renderer for free. This means that an artist can instantly see the result of their work with approximate lighting and shading without having to wait on renders. Even the fastest IPR rendering solutions still have a delay while the viewport resolves.

It was this aspect that made the toolset vital for use in *A Wrinkle In Time* as it allowed the team to significantly speed up the iteration cycle. We built a playblast style workflow into the editor that would move through the shot camera frame by frame and save out a screenshot from the engine's game render view. In many cases, the quality of these playblasts was high enough to send directly to compositing, and then to the client for creative approval. We estimate that this workflow saved thousands of render farm hours at the very minimum.

Using the tools available also significantly reduced the time needed to do the layouts and set-dressing in the first place. Even large layouts would often take less than an hour to achieve from scratch, and changes were even faster.

Between the real-time rendering, and the tools for rapid layout, we saw iteration cycles reduced to hours instead of days on very large and complex shots.

5 FUTURE DEVELOPMENTS

The combination of built in engine tools (such as foliage paint) and custom tools built to order using visual scripting allowed us to deliver bigger, more complex and more art directed shots than ever before. However, we also ran into some limitations, particularly with the built-in paint tools, and with manual edits to the spline scatterers. It became clear over the course of these two shows that what we really needed was a tool that was a hybrid of the two techniques - a rule-based procedural scatter that could be interactively painted into the scene, could be edited at any time, and that would retain these edits even if the scatter is reseeded or the rules are re-run.

We have therefore developed a second generation of our toolset, combining the best of both. It's called 'Diorama', and an early version has already been used very successfully on *Aquaman*. We hope to have the opportunity to present our developments at a future date.

ACKNOWLEDGMENTS

To Alan Bucior for his invaluable assistance with lower level 3D math and as a wonderful sounding board with years of experience in game development practices.