Rapid, High Quality Dailies with RenderFlow for *The Jungle Book*

Jared Auty Marlène Chazot Ruben D. Hernandez Marco Romeo
MPC*

Figure 1: *A comparative image of a shot from The Jungle Book. The preview (left), rendered with RenderFlow, gives a good understanding of shot composition, lighting/shadows, fur and depth of field for the finalized render (right).*

Abstract

While working on the visual effects for *The Jungle Book*, we had to produce high quality daily previews for internal and client reviews. These included fur, complex environments and many animated characters. To do this efficiently, we used *RenderFlow*, a pipeline-aware rendering API developed at MPC to build complex scenes, render elements separately and compose them together [Romeo et al. 2015]. In this paper we describe how the system was used in production and discuss performance evaluation over five shots from the film to assess the validity of the approach.

Keywords: visual effects, automation, rendering

Concepts: •**Computing methodologies** → **Computer graphics;** •**Information systems** → **Multimedia information systems;**

1 RenderFlow in a Nutshell

RenderFlow [Romeo et al. 2015] is a system developed to ease the process of creating previews for visual effects dailies, increase their quality and reduce the time required to produce them. In addition, the system is designed to enable technical directors to develop tools that take advantage of its capabilities.

RenderFlow gathers shot information from our digital asset management system and tracks which parts of it need to be rendered and which can be reused from a previous render. This saves considerable processing time needed to create a preview. To achieve this, the renders are split into their components (normally a single asset per layer), then composited back together, using their depth information, to produce the final daily. To achieve inter-layer shadowing, RenderFlow can define specific assets to be shadow-receivers and ensure that desired shadow-casting assets are used in rendering their shadows. This was particularly useful for generating contact shadows between characters and the ground.

RenderFlow is built as a node based Python API. Each node can represent a certain process to be applied on production data. Standard processes include building a three-dimensional scene, rendering it and compositing the resulting layers to obtain the desired result. Each node can execute processes in various third party applications (e.g. Katana) and then pass the result to the next node. This modular system provides flexibility to create workflows that are suited to the particular needs of a department. Furthermore this allows us to easily integrate and use different third party applications.

2 Animation Dailies

Due to the large amount of animated characters in *The Jungle Book*, rendering scenes with hardware-based methods was not a feasible solution. In fact, loading and displaying many complex characters and environments in one three-dimensional scene saturated the memory and took too much time. This produced flickering hardware renders, with disappearing geometries and multiple crashes.

Thanks to RenderFlow we were able to load and render each asset separately, which helped to improve the usage of our resources. Furthermore, with animators editing the animation of only a few characters at a time, it was possible to obtain a new finalized daily in a short time, because of the re-use of previously rendered assets.

Because at MPC we render deformable geometries in Katana using geometry caches, animators cannot start to render their previews until the caches are computed. As the generation of geometry caches takes time to complete, this could potentially cause delays. To solve this, we implemented the ability for RenderFlow to render an animated asset using Mental Ray by loading the character rigs and their animation in Maya, without the need to wait for the caches to be generated. The system would then composite the Mental Ray renders together with those produced with Katana.

Finally, to enable animators to choose the output quality and determine the rendered daily, we developed a tool that gave control over various features such as textures, fur, motion blur and depth of field. Each set of features could be stored in a preset, that was then easily accessible and re-usable by artists to ensure consistent results.

*e-mail:(jared-a, marlene-c, ruben-dh, marco-ro)@moving-picture.com

SIGGRAPH 2016 Talks, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-4282-7/16/07
DOI: http://dx.doi.org/10.1145/2897839.2927415

3 Quality Control

In production, quality control is key to improve efficiency. To properly review daily work and check that all assets have received the required updates, RenderFlow was used to produce simple renders that provided useful insight on the rendered content. By generating and storing extra data into custom image passes, we enabled supervisors to analyze the preview in detail. The passes that we implemented included:

- Level of detail (Figure 2): to review scene optimization;
- Instance sources: to identify instanced geometries;
- Wireframe: to give a hint of the resolution of meshes.

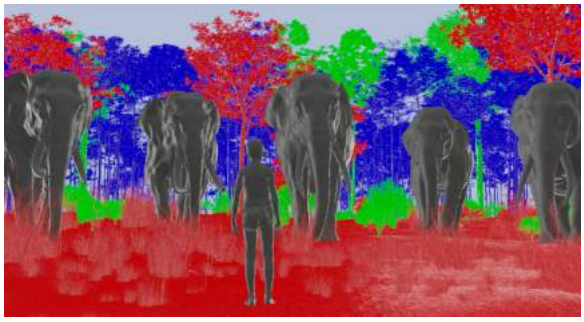

Figure 2: *Levels of detail (LOD) for various assets in a shot: A (red), B (green) and C (blue).*

With a film of such a large scale, it was also important to properly track the progression of production towards completion. To enable supervisors to assess progression, we prepared RenderFlow to collaborate with the MPC automated (nightly) assets update system, which updated shots if assets were modified. Once a shot was updated, a RenderFlow process was automatically triggered to generate a new daily for later review. This approach helped to rapidly identify pipeline issues or outdated asset versions being mistakenly used.

4 Results

To evaluate the approach of rendering and reusing assets in separated layers, we ran an experiment on five shots of the film, ensuring the same machine specification was used across all tests. The shots were selected to have a different number of assets between each other, ranging from 3 to 52. Each shot was rendered every time any asset was modified in production. Renders were performed two ways, by rendering all assets together (single layer) and by using different layers for each asset. In the most complex shot we tested (52 assets), we were not able to render all the assets at once, with the rendering process failing at 75% of completion, running out of memory on a 64GB machine; the same render was successful with the RenderFlow's layered approach on a 32GB machine. In the remaining four cases, RenderFlow showed a big improvement in rendering speed. Figure 3 graphically shows those results for the four shots.

Finally, the render obtained with RenderFlow in Figure 1 (left) took 54 minutes per frame to process (load, render and compose each asset) the first time. After that, any changes to a complex character required a maximum of 9 minutes per frame to re-render. This means that a new daily with the desired changes on one character could be produced in just 16% of the initial time.

5 Future Work

Future work will focus on enabling deep compositing and improving the current inter-layer shadowing approach. Furthermore, with the objective of maximizing the performance of the renderfarm, we will work to automatically define optimal sets of render layers.

Acknowledgements

We would like to thank Damien Fagnou who initiated the RenderFlow project and designed its concept. We would also like to thank Greg Fisher and Elliot Newman for believing in the project and contributing to its success. To Hannes Ricklefs and Rob Pieké for helping with development direction. Finally, we would like to thank Adam Cheshire for providing his lighting expertise.

References

ROMEO, M., AUTY, J., AND FAGNOU, D. 2015. Intelligent rendering of dailies: Automation, layering and reuse of rendered assets. In *Proceedings of the 12th European Conference on Visual Media Production*, ACM, New York, NY, USA, CVMP '15, 15:1–15:2.

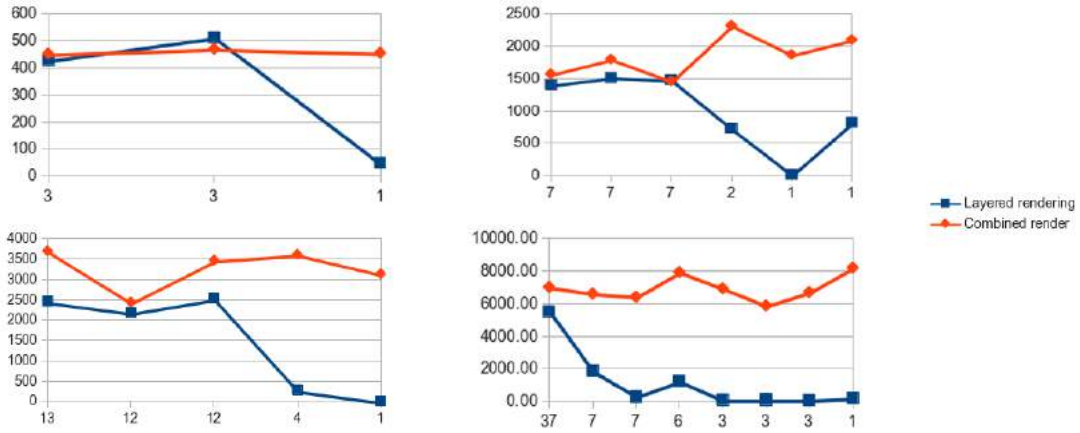

Figure 3: *Performance evaluation on four shots with different number of assets requiring re-render. For each graph, the vertical axis shows the render time (seconds per frame) and the horizontal axis is the number of assets being modified prior to re-render.*