Distributed Multi-Context Interactive Rendering

Alex Gerveshi
DreamWorks Animation

Sean Looper
DreamWorks Animation

Figure 1: Six shots from *How To Train Your Dragon: The Hidden World* rendered simultaneously in the cloud

ABSTRACT

By enabling artists to work interactively with renders of multiple shots from a single application, new lighting and surfacing workflows were made possible. This technique was implemented by replacing Katana's interactive-render mechanism, and by leveraging Arras, DreamWorks' in-house cloud computation system.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**; • **Computer systems organization** → **Cloud computing**;

KEYWORDS

multi-shot, multi-context, interactive rendering, cloud computing

ACM Reference Format:
Alex Gerveshi and Sean Looper. 2019. Distributed Multi-Context Interactive Rendering. In *Proceedings of SIGGRAPH 2019*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/8888888.7777777

1 INTRODUCTION

Interactive rendering is where user-applied modifications to a scene (such as moving a camera, increasing the intensity of a light, or changing a material property) are sent to the renderer and processed without incurring the cost of a full scene rebuild. This allows artists to iterate on their scene at interactive framerates.

When a lighting artist is working on a shot and makes a change to a sequence-level lighting rig, they will often want to see how the change affects the other shots in the sequence. For longer shots, the artist will also want to see how changes made in one frame impact the others. One way to accomplish this would be to send a number of key frames to the render farm, and then view the results some time later. If the time-to-first-pixel is short enough, they could render the frames locally in succession. Alternatively, if the lighting tool and renderer support it, they can change the frame and shot during an interactive render and flip back-and-forth.

None of these solutions are ideal; the turnaround time between making a wide-sweeping change and viewing the results of it prevents rapid iteration. It would be much faster to have each shot or key frame running as an interactive render within the same application, and then see a single change to the scene be propagated to each render.

There are two main challenges to achieving this. First is the limits of the artist's machine; if rendered locally, each shot would be competing for memory and CPU resources. This would reduce the possible complexity of the scene and slow down the interactivity of the renders as each renderer instance competes for CPU cores. For multi-shot workflows to scale, all expensive work needs to be moved off of the local machine. The second obstacle is making the user application capable of handling multiple active interactive renders; the value of multi-shot renders is greatly reduced if sequence-level changes cannot be applied simultaneously.

1.1 Multi-Context

Figure 2: Multi-context rendering of Poppy from *Trolls* with different lighting setups

In addition to working on different shots or frames, there are also use cases for working on variations of the same scene such as trying out different lighting setups or material networks. At DreamWorks, the term "multi-context" encompasses these workflows as well as multi-shot.

2 ARRAS

Arras is a cloud-based computation system developed at DreamWorks. An Arras computation receives messages, performs work based on their contents, and then sends outgoing messages. A group of computation instances passing messages between each other is referred to as a session.

MoonRay, DreamWorks' renderer [Lee et al. 2017], is designed to be Arras-aware. If an Arras session contains multiple render computations (with each computation running on a separate Arras machine), then the MoonRay instance in each computation will render different parts of the frame, with the results being merged together before they are returned to the client application. The time-to-first-pixel is roughly the same as for a local render, as each computation must build the entire scene. However, the time in which the render converges decreases linearly with the number of render computations in use; using 10 render computations will result in the render completing 10 times faster than using a single computation.

If the entire process of loading, building, and rendering a scene can be offloaded to Arras, then in a multi-context scenario the client application would only responsible for managing the Arras sessions and receiving the rendered frames.

3 KATANA

When a render is started from the Katana node graph, an optree is generated. The optree describes a sequence of operations that when evaluated will generate the scene graph. All loading of data is deferred: no scene data is loaded until the scene graph is expanded by Geolib3, Katana's scene graph processing engine. For a multi-context setup, this means that an optree for each context can be constructed in a single Katana session without having to load the scene data (such as a USD stage) for each shot. An Arras computation was developed that receives a message containing the optree, evaluates the scene graph using Geolib3, and then starts a MoonRay render.

The "ContextSwitch" Katana node was created to allow users to define multiple contexts, where each input to the node is a new context. This gives the user the flexibility to have separate node graphs feeding into each context, or using a single node graph that branches off to allow for context-specific variations. When the user starts a render from this node, the initial optree is generated for each context and an Arras session is started. The resulting rendered frame data is then copied to different sections of the Katana monitor based on a user-described layout.

The interactive render mechanism shipped with Katana requires the user to select the scene graph locations that they want to monitor for updates. As they make changes to the node graph, the optree is updated and the selected locations are re-evaluated by Geolib3. Updated location data is then streamed to the renderer. This mechanism has the following drawbacks: a) it is only capable of handling a single context, b) it requires the data for each scene to be loaded on the local machine, and is therefore unsuitable for multi-context interactive rendering, c) monitoring multiple scenes for updates would be computationally prohibitive.

Instead, a new system was developed that regenerates the optree for each context when the user makes a change in the node graph. This is a lightweight operation as Katana caches the ops associated with each node. The latest optree for a context is compared with the previous, and the difference between the two, the optree delta, is sent to the corresponding Arras session. There, the entire scene graph is being monitored for changes and the render is updated. This allows for each context to have different scene graph layouts and not have to rely on the user to specify what scene graph locations they are interested in monitoring.

With this method the local application is only responsible for generating optree deltas and receiving rendered frame data from Arras. All of the heavy lifting has been offloaded to the cloud. The user is only limited by the amount of cloud resources available to them. Each context can fully utilize the memory and CPU of the cloud machines assigned to the Arras session, rather than having to compete for resources on the local machine.

This technique is not limited to Katana, in theory any application that can construct an optree and connect to Arras now has access to multi-context rendering.

4 CONCLUSIONS

One of the main use-cases for multi-context at DreamWorks has been in interactive reviews with creative supervisors. Previously, when given notes on a sequence, the artists would then have to make the changes once the session was over, submit the updated shots to the farm, and then repeat the process at a future review. Now, using a multi-shot Arras setup, artists can make changes in the lighting tool with the director still in the room. These changes are not limited to lighting; changes to animation, geometry, materials, or any other part of the scene can be made while maintaining interactive frame rates. This scenario is even more compelling when each Arras session is using multiple render computations. For example, the artist driving the session can start a render with six contexts where the Arras session for each context requests four 48-core machines. To the users it then appears they have started six 192-core interactive renders from a single application. By reducing the necessity of offline rendering, the decision to "final" a shot can now be a purely creative one, rather than being subject to the time constraints of previous workflows.

ACKNOWLEDGMENTS

The authors would like to thank Stefan Habel and Jordan Thistlewood at Foundry, and Toshi Kato, Rob Wilson and the lighting tools team at DreamWorks.

REFERENCES

Mark Lee, Brian Green, Feng Xie, and Eric Tabellion. 2017. Vectorized Production Path Tracing. In *Proceedings of High Performance Graphics (HPG '17)*. ACM, New York, NY, USA, Article 10, 11 pages. DOI: http://dx.doi.org/10.1145/3105762.3105768