The Jungle Book: Art-Directing Procedural Scatters in Rich Environments

Stefano Cieri* MPC
Adriano Muraca† MPC
Alexander Schwank‡ MPC
Filippo Preti§ MPC
Tony Micilotta¶ MPC

Figure 1: *Scatter elements highlighted (left): static plants - red, simulated plants - green, tree flowers - yellow, debris - blue; final render (right) ©2016 Walt Disney Pictures. All rights reserved.*

Abstract

Disney's live action remake of *The Jungle Book* required us to build photorealistic, organic and complex jungle environments. We developed a geometry distributor with which artists could dress a large number of very diverse CG sets. It was used in over 800 shots to scatter elements, ranging from debris to entire trees. Per object attributes were configurable and the distribution was driven by procedural shaders and custom maps.

This paper describes how the system worked and demonstrates the efficiency and effectiveness of the workflows which originated from it. We will present a number of scenarios where this pipelined, semi-procedural strategy for set dressing was crucial to the creation of high-quality environments.

Keywords: environments, distribution, particles, instancing

Concepts: •**Computing methodologies** → **Computer graphics;** •**Applied computing** → **Media arts;**

1 Enabling Complex Scatters

The scattering system was conceived in the early stages of MPC's work for *The Jungle Book*, where we realized that environment artists needed a solution to populate scenes quickly and easily with great detail and varying scale. Before starting the development, a broad scope investigation was conducted, looking both at the research field and at available commercial, open source and out-of-the-box solutions.

1.1 Related Work

A variety of commercial packages and known techniques from other facilities were evaluated for adoption. An extensive collection of existing approaches is presented by Bradbury [2015], which focuses on vegetation but presents techniques also applicable to generalized scenarios.

The most common strategies involve procedural methods and point-sampling approaches: these are more suitable for context-agnostic distributions (with patterns not strictly related to the nature of the elements being scattered) but they mostly lack flexibility with regards to creative control and manual adjustments.

Simulations of natural phenomena and complex systems often deliver the most realistic and visually rich results, however they require tailoring to specific behaviours: plant and tree growth is shaped by a set of biotic and abiotic factors, which are radically different from those determining the distribution and accumulation patterns for dirt or debris. Also, there was a lack of documentation proving these methods reliable in production scenarios.

A significantly different approach was shown in *Wonder Moss* by Inigo Quilez [2014]. The technique was developed at Pixar for the movie *Brave*, to cover sets with plants, moss and minute details, directly at render time. This technique is particularly interesting as it works as a layer, wrapping any renderable geometry and seamlessly blending heterogeneous elements with different scales. Despite its effectiveness, the drawback of such an approach is that it requires an experienced programmer to address visual feedback, by tweaking the code of the render procedurals.

Established production and out-of-the-box tools (ie. Houdini, Clarisse, XGen) generally leverage some of the above methods and

*e-mail:stefano-c@moving-picture.com
†e-mail:adriano-m@moving-picture.com
‡e-mail:alexander-sc@moving-picture.com
§e-mail:filippo-p@moving-picture.com
¶e-mail:tony-mi@moving-picture.com

DigiPro '16, July 23-23, 2016, Anaheim, CA, USA
ISBN: 978-1-4503-4429-6/16/07
DOI: http://dx.doi.org/10.1145/2947688.2947692

provide efficient solutions. However, since integration with MPC's inter-departmental workflow was required, it was preferrable to develop a custom toolset in Maya (see section 2).

1.2 Chosen Strategy

Our system was designed to combine the flexibility of procedural distributions and the granular control required by art-direction. Its scattering principles aimed to maintain the highest possible level of abstraction and generalization, by following a stochastic approach where one could define the parameters computed by the logic layer. Depending on the desired scatter behaviour, artists would translate distribution rules into shaders and maps while controlling randomization via numeric parameters. Naturally, artists started developing their own techniques and strategies (with TDs building their own extensions and scripts) to deliver the required visual complexity.

2 Implementation

The main application for environment artists at MPC is Autodesk Maya, so we leveraged its established API and integration in MPC's pipeline to reduce development time while increasing the number of code iterations. This let us be agile and reactive in addressing artists' continuous requests for new features.

While Maya was used as an interactive front-end, similarly to Antoine and Allen [2004], an abstract logic layer was developed to handle the scatters and let artists configure them, with the release process transparent to them. To reduce the artist machine workload, the release process was distributed on the farm with a cut down version of the Maya scene.

2.1 Maya Integration

The core of the system consisted of a custom Maya node called *ptcFilter*. This node wrapped the scatter logic and output the result as particles with instancing attributes, by intercepting a geometry emitter, computing the attributes for every particle and using shaders to filter them. The result was then injected into a Maya nParticle system for visualization and further manipulation.

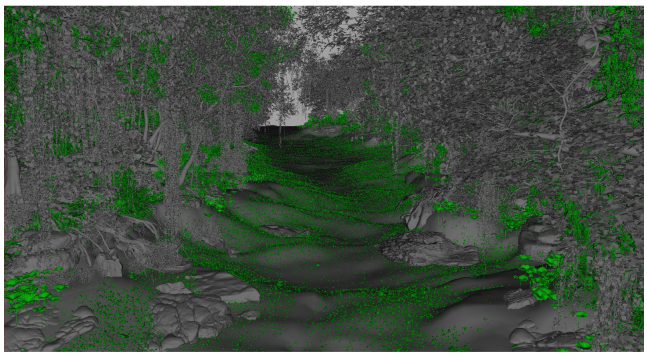

Figure 2: *Multiple scatter setups visualized in Maya - ©2016 Walt Disney Pictures. All rights reserved.*

If a scatter was approved for the overall look but required adjustments, artists could maintain the desired attributes and selectively re-compute others on any subset of particles. In this case, the computation was handled entirely by the logic layer, without involving the *ptcFilter*. When settings changed for any setup, Maya nodes' attributes and connections were updated accordingly. This provided an implicit serialization of the scatter setups in the Maya scene, which was crucial for a simplified management of the numerous scatter variations required by the show.

In addition, a Maya command based on MPC's Crowd technology allowed artists to paint particle instances on geometry by tracing camera rays onto it, while still applying distribution rules.

2.2 Abstract Logic

The abstract logic layer was designed to store and process the scatter parameters, per instance source, and to organize scatters in setups (when convenient for lookdev, lighting and rendering or when instances shared the same distribution pattern). The system was designed to manage multiple setups within the same session, giving the ability to tweak and store settings for each setup.

Scatter instances could be randomized programmatically: *Orientation* would still be affected by scene geometry (using normals) and weighted with an aim vector; *Scale* could also be set to have a random average for the main distribution and abnormal sizes. Additionally, *Occurrence* could be weighted to determine how frequently each asset would appear in the scene.

2.3 Pipeline

The tool was integrated into MPC's pipeline in order to output renderable packages that could be used by other departments as well. Scatters with a high particle count were released as particle cache collections - called *InstancePackages* - which ensured optimal render performances. Other scatters were released as *ModelHierarchyPackages* which allowed more specific manipulation and supported simulation.

In addition to dynamic simulations, the integration with MPC's grooming software *Furtility* allowed artists to grow moss and grass from scattered particles. Moreover, such particles were used by a custom PRMan shader, to generate 3D textures for the Lookdev department.

3 Workflow

The implemented features were available to the artists for them to use and combine freely. Building their own shading networks in Maya, they could use 3D textures or camera-based projections to control the look of their scatters. Typically, shaders were used to control the density of distributions (ie. to produce denser scatters in interstices, using *Ambient Occlusion*) and to drive the behaviour of instances on slopes or to achieve bespoke looks with painted maps. Instances could also be painted in viewport, directly on geometry, with a few brush options.

While the main workflow was already used in production, an additional technique was developed to generate instances via PRMan renders; a custom REYES shader was written to generate point clouds where scatter would occur. This way, we were able to accurately compute the geometry displacement and to store the terrain's texture values as primvars, which were then used at render time to drive color variation and help blending with the underlying geometry. Optionally, bent normals (representing the average unoccluded direction) could be computed to adjust the instances' orientation organically. Lights could also be placed to represent the wind direction and let scatters accumulate in shadowed areas. Finally, the point cloud was converted into Maya particles and hooked to the abstract logic, for the artist to operate further on them if required.

3.1 Artist Efficiency

Any set of geometry could be scattered, independent of its polycount. Due to the procedural and flexible nature of the system, artists could easily reapply the scatters to the numerous environment iterations, while still being able to make per object adjustments.

Since the system output native Maya entities, it allowed particle selection at component level to make adjustments on any subset of them. Scatters could be visualized both in Viewport 1.0 to support MPC legacy nodes, and 2.0 for improved performance on particularly heavy scenes. In addition to Maya, artists were able to preview their work in Katana. In both applications, scatters were presented as particles, bounding boxes or actual models depending on the use case.

On average, a scatter for simple scenarios could be set up and ready for rendering within a single day. Full sets typically took one to two weeks due to their large scale. A typical scatter setup consisted of a few thousand to a few million elements; often regardless of the scale of the framed set (close shots required denser and smaller detail). In total, a single shot easily contained up to ten million scattered elements.

4 Future Work

The main future objective of the system is to keep improving performance of scatter sessions. For attribute computation, we plan to solve the inefficiencies we found in certain Maya API calls, with lower-level manipulation of the particle nodes and memory. For visualization, we intend to use Fabric Engine [Fabric Engine 2016] in both Maya and a standalone tool, to give previews of scatters instancing high-resolution models.

When rendering, we knew that heavy instancing of high-poly meshes would be the best way to preserve detail in the foreground with minimal memory footprint. During production, we found that excessive detail in the background elements would produce noise which would make renders take longer. To help optimize the trade-off between render time and memory, we plan to add a level of detail management for scatters' instance sources.

Acknowledgements

We thank Inigo Quilez and Alan Stanzione for inspirational code, research and enriching chats; Marco Genovesi for letting a simple prototype become a proper production tool; our kind supervisors, Marco Rolandi, Daniele Bigi, Audrey Ferrara, Elliot Newman and Adam Valdez for backing the idea; Fanny Chaleon for the integration with *Furtility*.

A big thank you goes to all our great colleagues who gave suggestions, feedback and support, especially: Dora Morolli, Luca Bonatti, Patrick Hecht, Thomas Wolstenholme, Piotr Szuter, Robin Huffer, Mark Laszlo, Clair Bellens and Tom Melson.

They all really made the difference.

References

ANTOINE, F., AND ALLEN, D. 2004. Leveraging third-party tools for art-driven fluids & foliage. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, 72–.

BRADBURY, G. A., SUBR, K., KONIARIS, C., MITCHELL, K., AND WEYRICH, T. 2015. Guided ecological simulation for artistic editing of plant distributions in natural scenes. *Journal of Computer Graphics Techniques (JCGT) 4*, 4 (Nov.), 28–53.

FABRIC ENGINE, 2016. http://www.fabricengine.com/.

INIGO QUILEZ, 2014. Wonder Moss. http://youtu.be/Z_Vk3Yn-wCk. Accessed: 2016-06-03.