Using *Cortex* Open Source to build the Crowd Pipeline for *Happy Feet Two*

Carsten Kolve Rogier Fransen Moe El Ali Daniele Niero Dan Bethell
Dr. D Studios

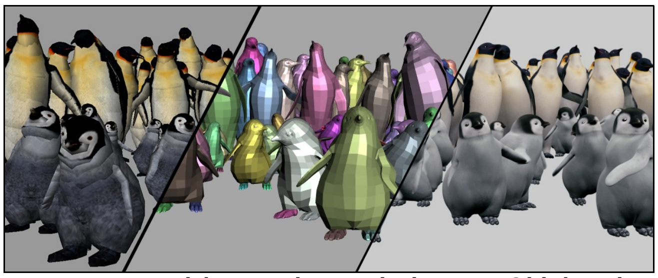
Figure 1: Maya crowd shape, Houdini procedural geometry, 3delight render – all utilising identical Cortex driven backend functionality ©Dr.D Studios Pty Ltd. All rights reserved.

Abstract

For *Happy Feet Two* a team of 3 developers built a crowd pipeline able to produce a variety of crowd effects for over 620 shots with a 7 month main production period and a team of average 16 artists. The crowd workflow interfaced with nearly all production departments, from previs all the way to lighting, and worked across multiple platforms.

The main focus of this sketch is not crowd generation; techniques for this have been covered in numerous previous works - instead it is a case study of using the open source software library *Cortex*[1] as a base foundation for rapid development of custom software solutions, like a crowd system, in a VFX / feature animation context.

1 Choosing Cortex

After having completed crowd workflow specifications it became apparent that implementing all required features within the given 8 month timeframe was not possible given the available manpower. After evaluating the open source library Cortex for its suitability as a foundation of the crowd system, we anticipated being able to reduce the development time by over 30% compared to implementing a complete custom in-house solution. This allowed us not only to stay within our resource allocation but also profit from extra functionality not budgeted for.

2 Using Cortex

Dr. D's crowd system was based on the concept of mass instancing pre-recorded skeletal animation, typically acquired by motion capturing. For 95% of all shots no simulation techniques at all were used – instead we relied on artists to manually and procedurally place crowd characters in a shot. The resulting layouts were then passed to other departments for further processing. Based on these layouts crowd character components were assembled on demand.

Cortex formed the back-end for all crowd related data and operations. Following a breakdown of the main components of Dr. D's crowd pipeline and how they utilised Cortex.

Core Data Types and Operations

Character and terrain meshes are exported from Maya and stored using Cortex' MeshPrimitive type. For animation data we store arrays of matrices to describe skeletal poses. An important addition to the already existing primitive types in Cortex is a skeleton representation: Cortex' serializable CompoundObject classes, which are similar to runtime assembled C structs, allowed us to prototype a suitable structure in Python, before implementing it as a SkeletonPrimitive in C++.

Together with Image Engine VFX, the initial developers and main contributors to Cortex, data types and operations were put in place to handle bind-information for smooth skinning type deformations. With these small additions to Cortex all elements were in place to animate a skeleton and deform a geometry bound to it procedurally.

Data Management

Our data needed to be organised in ways that make it easy and efficient to access and store: Animation data is stored in a hierarchical structure that allows for fast sequential as well as random access of multiple animation poses. The implementation is based on the FileIndexedIO interface provided by Cortex. For faster repeated access, mainly utilised during interactive playback, we used the existing LRU caching module. This enabled us to access minutes of unique animation data for hundreds of characters at interactive rates.

All character related components were stored in custom CompoundObjects optimised for convenient access. These structures were built using small Python scripts that analysed asset structures prior to conversion. Layout information which references animations, character and terrains was also stored in a similar structure and accessed through a custom function-set allowing higher level operations such as merging of layouts.

Manual and Procedural Crowd Layouts

While in Maya, crowd layout data was passed through Cortex Python scripted operator nodes that procedurally modify the data, enabling operations like clumping, terrain adaption or collision resolving. A custom Maya shape where each component represented a single character visualised through articulated cut-up geometry was used for manual layout. For speed reasons a specialised OpenGl display solution was preferred to. the Cortex generic Gl rendering module.

Crowds that required a more procedural approach, like flocks and swarms, were created in Houdini. These setups utilized the same ops in use on the Maya side, but this time interfacing with native Houdini technology. Op scripts were always the same, no matter the platform, allowing for quick reuse in multiple contexts – even in standalone Python.

Rendering & FX

For HF2 all lighting and fx were done in Houdini, all rendering in 3delight. For crowd rendering two Cortex Python scripted delayed Renderman procedurals were used: The first one determined character bounding boxes and injected sub-procedurals responsible for generating the final character data at a given lod at rendertime.

The same operations generating character geometry during rendering were also used to generate temporary geometry at runtime in Houdini, used by the fx department to generate secondary effects like foot steps and snow kicks. Integration of Cortex in Houdini was co-developed with Image Engine.

3 Conclusion

Even though the initially anticipated functionality of Dr. D's crowd system differed in many aspects from the final toolset, Cortex has proven to be a flexible base for general VFX software development and a main contributing factor for delivering the pipeline in time.

Due to its modular structure, a broad set of existing features and the fact that most of its functionality is readily available in Python we believe it is well suited for rapid prototyping. Extending Cortex directly did require a longer familiarisation period, but using both the Python or C++ libraries was straightforward.

Participating in the open source project development and committing code back has enabled us to receive feedback and bug fixes from the community, foster good coding practices and hence develop higher quality code faster. Confronted with other R&D tasks in the future, we would not hesitate utilising Cortex again.

1 http://code.google.com/p/cortex-vfx/