Data-driven Background Crowds in *Exodus: Gods and Kings*

Martin Prazak Mungo Pay Damien Maupu Davide Vercelli Ian Masters
Double Negative Ltd.*

Figure 1: *Stills of background crowds in* Exodus: Gods and Kings. *All images* ©*2014 20th Century Fox.*

1 Introduction

Large virtual worlds require large virtual crowds. One of the main themes of *Exodus* is a clash of ancient civilisations, represented via virtual background crowds in a large number of shots. To generate these crowds, we used an in-house crowd solution, whose development was aimed at background crowds for photorealistic scenes.

VFX crowds are subject to specific requirements, because they are used to enliven and extend crowd shots captured by a camera. Each shot consists of three layers – *hero characters* (usually real actors shot on set), *crowd-anim* (hand-animated) and procedural *background crowds*. To retain scene consistency, the background crowds must join seamlessly with other layers.

2 Manual or Procedural?

Large numbers of background characters in crowd scenes necessitate some level of procedural generation. Traditional approaches (in tools like Massive or Houdini) use a combination of data-driven and procedural animation synthesis, steered by behavioural simulation (particles/boids, fuzzy logic). While per-agent simulation leads to a flexible and powerful simulation framework, small setup changes can lead to different and unpredictable outcomes. For a crowd artist, it can be frustrating if a minor tweak of a seemingly unrelated parameter causes large and unpredictable knock-on effects.

While our system includes a boid simulation module based on social forces, the most used functionality involves artist-driven vector fields, influence regions and keyframed behavioural switches with randomization. Data-driven behaviour has proven to be favoured by artists for its controllability and predictable results. Agent instantiation is handled in a similar way: with probability maps and conditions, the user can create a large number of characters with randomized properties, while retaining a tight control of special cases.

Motion synthesis utilises artist-provided animation clips and combines them using state machines and blend trees. Artists often provide tailor-made clips to achieve a particular look-and-feel, which would be hard to synthesize. Artists can also animate whole groups of characters, allowing complex agent interaction. Our system provides tools for instantiating these, and for tweaking their trajectories and time offsets to allow the artist to achieve the desired outcomes.

*E-mails: {map,mungo,dmu,dv,iim}@dneg.com

This approach leads to "crowd compositing" – creating large simulations by combining pre-simulated vignettes.

However, users expect some motion editing to be performed automatically (with the ability to override results if necessary). Firstly, during the setup stage of a motion state machine, possible transition points are detected automatically. Secondly, the animations have to be applicable to any similar character, making a semi-automatic retargetting procedure necessary. Finally, footstep cleanup and ground adaptation have to be performed for any locomotion clips.

3 Under the hood

From production experience, we have found out that off-the-shelf 3D software does not lend itself well to crowd requirements; our system is designed to address the most common shortcomings.

In the core of our system there is a flat shared memory data structure, created during simulation initialisation by collecting per-frame storage requirements of all nodes. Because the data layout and access requests are known beforehand, we can create a data dependency graph with parallel branches evaluated simultaneously. This, combined with the use of instanced geometry and shaders, allows us to display up to 100k agents in close-to-realtime.

4 User interaction

To facilitate a variety of workflows, we have developed a number of tools to ease the setup of scenes and drive simulation properties. These include a painting tool so that artists can paint on geometry to trigger behaviours, and Python integration for the more technical artists so that scene setups and agent behaviours can be scripted.

5 Rendering

The large number of characters in a scene makes the use of optimisation techniques necessary – a typical scene with characters represented as animated meshes would occupy 3 orders of magnitude more space than a corresponding bone-transform representation.

A classical approach of mesh instancing is not applicable to crowd scenes, because the probability of two characters of the same variant displaying the same frame of the same clip is very low. Our system makes use of 3 main techniques to tackle this problem. Firstly, we use a proprietary file format storing bone transformations to represent the simulation data (with undeformed mesh and skinning stored separately); the mesh deformation is evaluated in the renderer, saving a significant amount of disk space. Secondly, we use a "procedural" with lazy evaluation, i.e. each character is instantiated only when a ray passes through its bounding box. This makes sure that skinning of occluded characters and characters outside camera frustrum is not computed. Finally, we reuse most of look data, computing most of character variation procedurally.