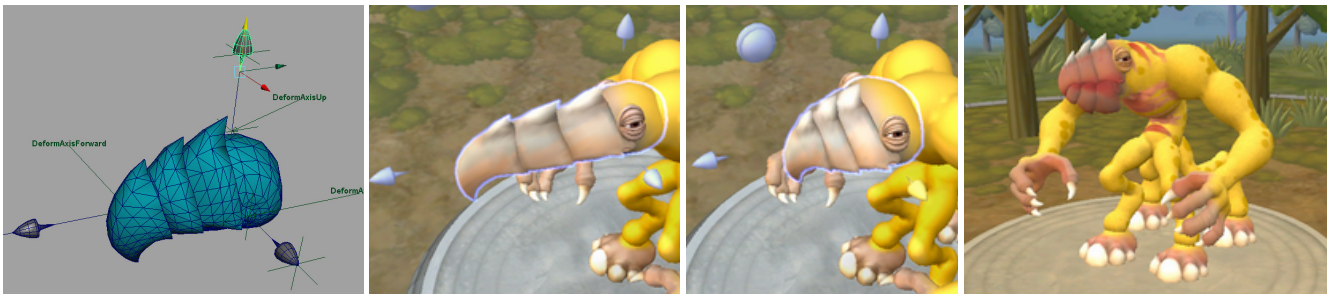
Rigblocks: Player-deformable Objects (sap_0248)

Lydia Choy Ryan Ingram Ocean Quigley Brian Sharp Andrew Willmott*
Maxis, Electronic Arts



From left to right: A Rigblock being authored, deformed by the player, and the final game model.

1 Introduction

We have built a system that allows the player to create various game models themselves, by assembling and deforming parts. This gives them compelling input into what their game world and avatar looks like. A key part of this system is what we term a *Rigblock*. A Rigblock is geometrical building block, representing a particular component of the model. For example, for a creature model the block may be a hand or mouth, for a vehicle, a wheel or jet engine. Physical analogues include Lego and mechano sets and their various parts.

A Rigblock is not static, however. In addition it contains a set of animations that can be used to deform the block in interesting visual ways. These animations are not time-based, but are rather parameterized over a unit interval, and are driven by one or more *handles* that the player controls. Handles are simple sliders, which scrub through the animation as they are moved from the start to the end of their range. They are displayed near to the feature they control, making it simple for the player to click and drag on the handle, and produce the desired deformation.

The advantages of our approach are:

- Player interaction with the block is intuitive and straightforward. The physical nature of grasping a handle and moving it back and forth to modify the block works well.
- Rigblock deformations are expressive. Because the corresponding animation is not constant-rate (the player can scrub through it at will), a number of variations are possible, from simple handles that linearly control feature size, to handles where subranges result in completely different styles of block.
- Rigblocks provide a balancing mechanism between enabling player creativity and amplifying player creativity. They fall in the sweet spot between high-quality artist-created models, with no player control, and lower-quality, effort-intensive, wholly player-driven approaches, such as providing a sculpting tool.

2 Pipeline and Workflow

A standard workflow for game animation is to have a separate author file per animation. These files in turn reference a standard skeleton. For Rigblocks it is crucial that the artist be able to view the composite results of applying multiple animations at once, hence a different workflow was required. To solve this problem we used Maya's non-linear animation editor. This required both reverse engineering some of its behaviours, and upgrading our pipeline to be able to handle exporting all animation tracks instead of just the main track.

We wrote MEL scripts to simplify Rigblock construction. Firstly the artist creates and places a handle, which in turn creates and connects the corresponding animation. The artist can then use the same workflow as in-game: scrubbing the 3D handle back and forth drives the corresponding animation via expressions. On export, the handle positions, ranges, and associated animation are propagated through the pipeline to game format.

Production proceeded by storyboarding sets of required parts for the game, and providing some example models that established style and a range of handle approaches. The artists then proceeded to fill in the story boards.

3 Animation Technology

When multiple deform animations are present, we need a method for composing them. The standard runtime approach is to cross-blend animations, with either per-animation or per-animation, per-bone weights. Deform animations require a different approach: we cannot weight them, as then applying a second deform would undo part of the effect of the first deform. Instead we want them to compose additively, in much the same manner as delta morph targets. (We also support morph targets for rigblock animations, but prefer to avoid them if possible because of high memory and runtime cost.)

The initial approach we took was to concatenate each animation in turn with the base pose inverted out of it. While this worked well, it did not match the results as previewed in Maya, which became an essential goal for artist convenience. Maya accumulates translation, scale, and rotation deltas from all animations individually, only forming the final transform matrix at the end; the engine was updated to take the same approach. (For rotation summation Maya uses Euler angles, whereas we concat quaternions; the difference was not large enough to matter. Although using euler angles is undesirable in a runtime engine, it does have the advantage that the result of applying N delta animations becomes order independent.)

Once a player-created model has been finished, it can be "baked" into its final form, and all animation data removed, in order to produce a model that can be rendered at game rates.

*e-mail: awillmott@maxis.com