dRig: An Artist-Friendly, Object-Oriented Approach to Rig Building

Greg Smith Mark McLaughlin Andy Lin Evan Goldberg Frank Hanner
Walt Disney Animation Studios

Figure 1: Characters created with the dRig system

1 Introduction

Ever increasing complexity and performance demands on animated characters in feature films necessitate unique rigging workflows and toolsets. We present dRig, a novel approach to rigging that allows for efficient reuse and extension of existing assets, fast authoring of per-element variations, and most importantly, accessibility of rig code to the entire department crew. By leveraging the powerful concepts of object-oriented programming and presenting it to the user through a well managed, artist-friendly interface, dRig provides a framework for sophisticated rig evolution and development.

2 Object-Oriented Approach

Unlike traditional procedural rig-building systems that automate the creation of element templates (faces, arms, spines, etc.), dRig is rig agnostic. Rather, it is a tool used to organize and modify the code used to create the rigs. Applying object-oriented design to rigging provides several benefits. These include encapsulation of a rig and its parts into reusable pieces and extension of any of these pieces by characters/types. These features allow the rigger to organize, override, and extend the individual features of each unique rig.

The rigs are defined by simple, human-readable text files that hold small units of code and variables. These variables are organized into hierarchical groupings that follow logical rig structures (e.g. body|left|arm|hand|finger). This hierarchy produces unique paths that the artist can use to easily navigate to the desired piece of the rig build. The small size of each block (and the code within) reduces the scope of a modification, allowing for more targeted adjustments. To ease debugging, these groupings are represented in dRig's interface as a tree view and the code/variables are represented in an expanded details panel. These views are akin to Maya's Outliner and Attribute Editor, which users are already familiar with.

In addition to the tree structures, dRig has the ability to layer these files. Each layer can mask the variables and code from the previous layer(s) by redefining the block at a given path. This allows for file A to inherit all of the information from file B while still allowing for surgical edits/additions to any aspect of the build. This inheritance allows for extensive reuse and customization from archetypal characters (bipeds, quadrupeds, etc.) or from other similar characters within a show.

It is significant to note that the result of "compiling" these files is not the final asset, but rather, an aggregate composite of code and variables that can be executed within our primary animation package (Maya) to create the final rig. This build process is managed through an intuitive interface that is akin to a programming IDE. It allows artists to execute and debug their code by setting break points, stepping through the build and getting instant feedback on errors or warnings in their rig code.

3 Artist Accessibility

One of the most important features of dRig within the rigging workflow at Disney is artist accessibility. Many rigging systems require highly technical users to create and augment rig pieces or the code that creates them. This limits quick iteration, innovation, and ultimately crew engagement. In contrast, dRig's simple, readable file format, its organization of code into navigable hierarchies of bite-sized code, and its intuitive interface allow even new artists and novice coders to debug and augment the rig setup.

To further open the code base to multi-user and multi-show editing, a git repository back-end is used to track edits made to the rig files. This allows for flexibility of individual productions to iterate and improve in an agile fashion without disrupting other productions that need stability in their rig builds. The shows can use standard git management tools to merge divergences and updates of rig definitions.

4 Evolution

By making the rigging code base more accessible, the rig builds see constant development and evolution. Through inheritance, organic, studio-wide standardization is realized as popular code blocks are layered/linked and propagated across rigs. Additionally, clear organization of code allows precise, incremental improvements in rig functionality as well as facilitates broader changes in base rig structures. A sophisticated rig and build procedure can evolve with the crew, and with each new show, to achieve new and demanding performances.