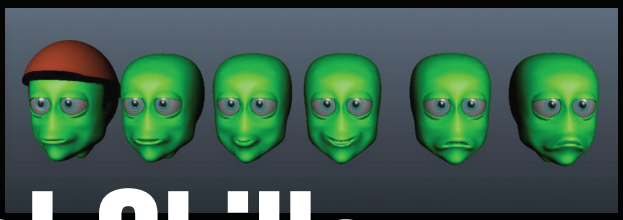
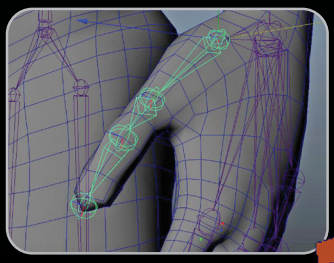
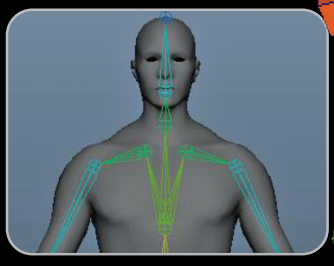
CRC Press
Taylor & Francis Group
AN A K PETERS BOOK

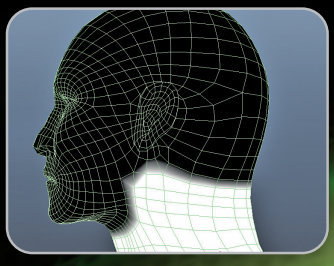


Essential Skills in Character Rigging

Nicholas B. Zeman







Essential Skills in Character Rigging

Essential Skills in Character Rigging

Nicholas B. Zeman

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20150827

International Standard Book Number-13: 978-1-4822-3524-1 (eBook - PDF)

Contents

Author

Nicholas Bernhardt Zeman started his career in 3D graphics at the University of Kentucky, Lexington, Kentucky, where during graduate school he began working in 3D Studio Max for the first time. Determined to make 3D graphics and games his career path, he left Kentucky for San Diego, where he was offered a job at Red Zone Interactive, a then-small company making the NFL Gameday series for Sony Computer Entertainment. He continued working for them as an expert in character rigging, facial rigging, and facial animation after they were purchased by SCEA until the team was disbanded and the NFL Gameday series was canceled after losing rights to EA for the football franchise. After that, he worked briefly as an animator and facial rigger for SCEA's motion capture and cinematic studio, working on SOCOM 3, among other titles. He was quickly hired by Take Two Interactive in San Rafael, where he continued to develop and manage character rigs on the NBA 2K series, All-Pro Football 2K8, MLB 2K9-10, and NHL 2K9. After almost 12 years in character rigging for sports games, he decided to leave the employment of game developers and focus on the academic pursuit of interactive development and became professor at Northern Kentucky University in the Media Informatics Department and began his own digital media technology company, RHZ Development LLC, where he continues to consult and produce functional games through gamification, mobile apps, and mobile games under the studio brand Little Fish Games and RHZ Development.

Introduction

In 1998, I was working at redzone interactive, a third-party publisher for Sony Computer Entertainment America. We were working on NFL GameDay for the Playstation 2, which hadn't actually been released yet. With deadlines coming up and a completely unknown landscape for the development technology, we were in a bind. We *knew* that the Playstation 2 could do so much more than the original Playstation, but we had no idea just how it would work or what it could do! Since I was the only person there with significant experience and time working with character animation using the new software, Maya 1.0, I was tasked with figuring out how we set up the character with all these new tools. Hence, the job of "character rigger for games" was born.

Character rigging had been going on for some time in the cinematic world, where software rendering allowed the artist to take advantage of tons of tools and techniques. But this was 1998—games could barely be displayed in 3D, much less process full character rigs. Usually, the tools, even with the fastest machines, were slow and extremely difficult to work with, requiring a lot of switches for preview optimization and then waiting for a long time for the renders to reveal what the end result would really look like. But in a game development environment, you don't have that kind of time. In fact, everything *has* to work in real time, *all* the time, from any camera angle.

The thing about character rigging, even in those early days, is that not a lot of people were interested in doing it. Most of the game artists were… *artists*: sculptors, figure artists, graphic artists, animators, and so on. Very few of them were keen on the technical aspects of 3D animation and characters, so it fell into the laps of the few technical artists who were out there waiting to solve puzzles and create systems within which the aesthetic art

could shine (probably because we weren't very good at the other aspects of art). One of the reasons why I took on this task is because (a) I couldn't draw worth a damn and (b) I secretly liked it. Technical art is a lot like solving a puzzle, but with a visual component to that solution.

As I delved into the process of character rigging, I started to peel back the layers of the things that define character rigging. Hierarchies, skeletons, rotations, deformation techniques, constraints, relationships, kinematics, and above all anatomy are all vital to the creation and maintenance of a good character rig. And these things are not easy to understand. Sometimes, you feel like you need a degree in human anatomy, computer science, and mathematics all at the same time! And that's not an unfair assessment. You really can't do character rigging without some degree of knowledge in all three of these areas.

So I, along with a lot of other talented individuals a lot smarter and more motivated than myself, began pioneering the technique of getting real-time character rigs to work in games. Along that road, I learned a lot about all the skills listed earlier, which go into the development of character rigs for animation and deformation. It wasn't an easy road—a lot of the knowledge that I was lacking had to come from outside sources. Basic coding, anatomy, and vector math were a few of the things I had to learn along the way in order to even be able to put together certain parts of a human-based character rig.

Due to the long hours of all the pioneering character riggers, from both cinematic and game development, many new "auto-rigging" solutions are popping up in multiple packages. These auto-rigging solutions seek to take all the repetitive work out of setting up a character for animation, and indeed they are tremendous time-savers in a lot of ordinary circumstances, when you need to animate or edit motion capture. Mixamo, Poser, and HumanIK are some examples of these auto-rigging systems available. The only disadvantage of these methods is in the "staleness" factor, where everything starts to look the same, as well as the limited functionality of the said rigging systems. A real professional rigger generally must create the rig from the ground up in order to work out every individual mechanic necessary for the proper creation of a character.

This book is not everything I know about character rigging, but it's everything you need to know in order to comprehend the multiple basic factors that go into setting up a proper rig for a 3D character model. I have taken 14 years of knowledge in this area and condensed it into the absolute essentials for you to learn and comprehend. After reading this book and

doing the exercises contained, you should be able to take any modeled character from any software package and construct a skeleton, bind and edit skin weights, and create a simple controllable rig for an animator to work with. The book relies heavily on Maya-specific tools; however, it is not absolutely tied to the software. Other packages have very similar tools (they have to), although implementation may differ from one to another. But the basics always apply.

I

Rotation

CHAPTER 1

Hierarchies

1.1 WHAT IS A HIERARCHY?

A hierarchy is a transform-based relationship between two or more objects, or transform nodes. One object is known as the "parent," and the other object is known as the "child." Two objects connected to the same parent are known as "siblings." You can have as many objects in a hierarchy as you wish, and parents can have multiple children, but a child object can only have one parent.

Hierarchies are very prominent in computer-based systems, because they are instrumental in categorizing and organizing data. It's also used in something much more commonly understood by the casual computer user—folders and files!

Folders and files are set up in a hierarchical manner, which is to say that you "put" files or folders into other files or folders, and when you move the folder at the top, the rest goes along with it! In this way, you can organize where stuff is and move it or copy it as a unit. Everyone who uses computers understands this system a little bit, or they wouldn't understand how to use the computer (although I've had several friends who just dump everything on their desktop and clutter the hard drive randomly). This is the core concept of "inheritance," which is what the folder analogy stands for. It simply means I can put an object or collection of objects into another object and move them all around from a single collection. Anything I do to the top-level folder gets applied to the rest of the folders and documents inside of it. For instance, if I "copy" a said folder, everything inside of it is also copied. If I "delete" a said folder, everything inside of it is also deleted (Figure 1.1).

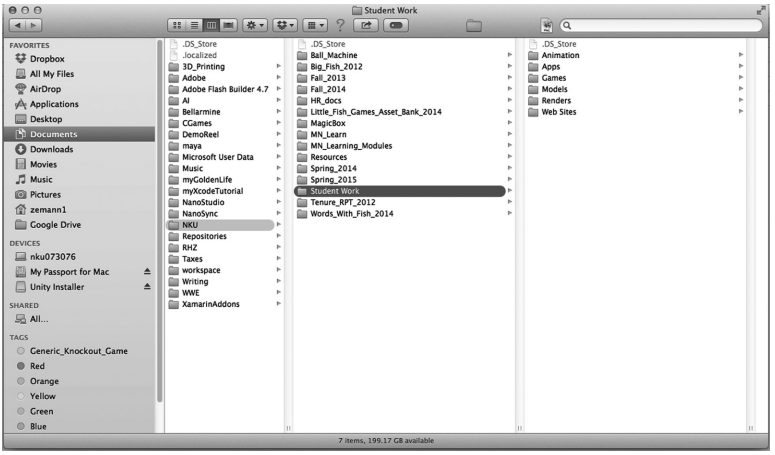


FIGURE 1.1 Windows/Mac folder structure.

1.2 HOW DO YOU MAKE A HIERARCHY IN 3D?

Constructing a hierarchy is actually pretty easy in any 3D graphics program. You select one object, and then another, and choose "parent" or link it in some way. The parent/child relationship is constructed in some manner by choosing the child, then the parent, and establishing the relationship. This is much like making a folder "Parent," in a computer filing system and placing the folder: "Child" inside it. The Child folder is now contained inside of the Parent folder. In much the same way, the child object's transforms in 3D are contained within the parent object's transforms (Figure 1.2).

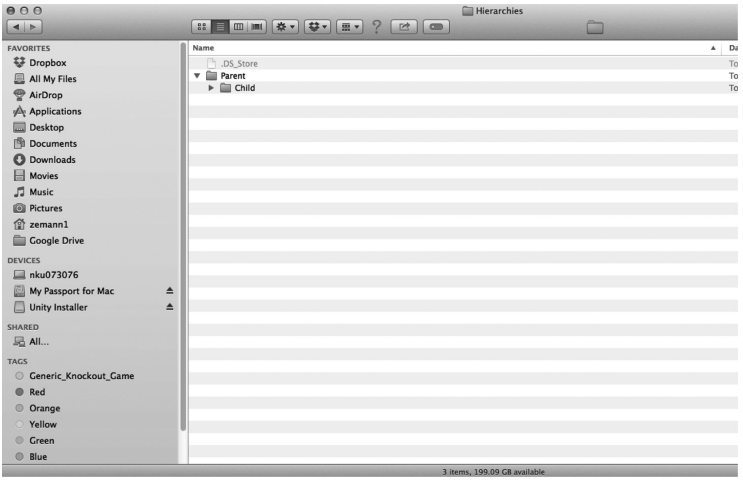


FIGURE 1.2 Parent and Child folders.

In Maya, you can see that when you select one object, that is, the parent of another, both objects are highlighted. This is done to indicate that you have selected an object with a hierarchy, and any objects beneath the selected object will inherit the transforms of the selected object. If you select the child object in this relationship, the parent object will not highlight and you can move it independently.

1.3 WHAT DOES "INHERITING TRANSFORM" MEAN?

1.3.1 How Does the Parent/Child Relationship Affect Global/Local/Object Transforms?

The relationship between parent and child in 3D is a complex one. Let's explore it a little. First, it is the *transform* that is affected by the relationship. This means that if you rotate, translate, or scale the parent object, the child will also be transformed as well. But the "child" object's transforms will not reflect value changes! In Figure 1.3, the cube object named "child" has a translate x value of 5 and the sphere object named "parent" has a translate x of 0; when you move the parent to translate x of 5, the child's transform x value in the channel box as shown in Figure 1.4 remains at 5 even though it moved 5 units to the right with its parent! Why is this?

In order to answer this question, we have to look closely at our old friends, local and global transformation. The values that you are seeing in the channel box in Figures 1.3 and 1.4 are *local* values. This has a special meaning in Maya (and all other 3D packages). *Local* values reflect the position of the

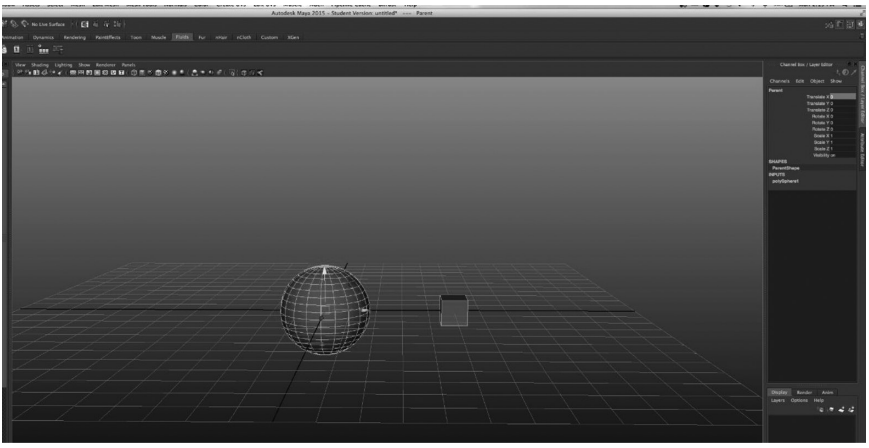


FIGURE 1.3 The sphere is the parent and the cube is the child. The child has a translate x value of 5 and the parent has a translate x value of 0.

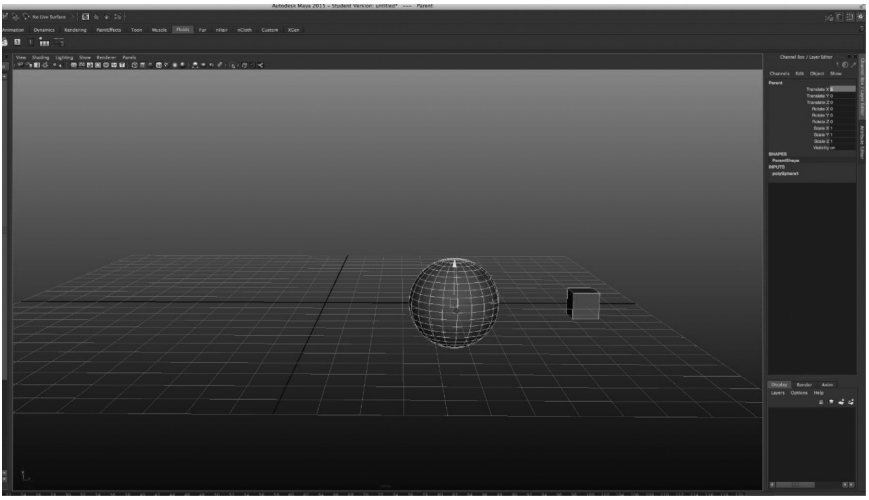


FIGURE 1.4 The parent object sphere has been moved to a position x value of 5, and the child object cube has inherited this transform. The local translate position x of the child object, however, remains at 5.

object *relative to its parent*. This means that its position value in the *x*-axis is still 5, even though its *global* location is actually the world x value of 10. No matter where you move the parent object in space, the child object will still have that value of 5! That value won't change until you move the child object itself. So the local transform value of the child object represents its offset from the *pivot* location (or transform center) of the parent object.

If an object has no parent, it is considered to be a child of "The World," which is basically a fancy way of saying it has no parent and all of its local transforms will be based on the global system. What that really means is that local values and global values will be the same if an object has no parent. Now, every object in a hierarchy can be parented to another, and chains can be formed, where transform inheritance keeps going down the branch—but I'll spare you that headache for now. Let's just get our head around this simple parent/child relationship.

OK, now that we (sort of) get how local and global transforms work with pure values, let us delve into coordinate systems and how they work with hierarchies. Figure 1.5 is the tool box from Maya for translation, which illustrates multiple *options* for translating an object. This is extremely important and misunderstood by many novices. In order to translate the object in question, we must first determine the coordinate system we intend to use. This only becomes really apparent when we *rotate the object*.