12-2015

THE DESIGN, IMPLEMENTATION, AND INCORPORATION OF "HACK AND SLASH" ANIMATION INTO THE UNREAL DEVELOPMENT KIT

Christopher Thomas
Clemson University, ct2@g.clemson.edu

Follow this and additional works at: http://tigerprints.clemson.edu/all_theses

Part of the Art and Design Commons

Recommended Citation

THE DESIGN, IMPLEMENTATION, AND INCORPORATION
OF “HACK AND SLASH” ANIMATION INTO
THE UNREAL DEVELOPMENT KIT

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Fine Arts
Digital Production Arts

by
Christopher R. Thomas
December 2015

Accepted by:
Dr. Brian A. Malloy, Committee Chair
Dr. Joshua Levine
Dr. Timothy Davis
Abstract

In this thesis, we investigate the incorporation of "hack and slash" game play animations into a first person shooter game engine, the UDK 4. We create the animations using the Maya modeling tool and we describe our approach to incorporating these animations into UDK 4 and create an appropriate control scheme to play these animations in a "hack and slash" game environment.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title Page</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td>List of Source Listings</td>
<td>viii</td>
</tr>
<tr>
<td><strong>1 Introduction and Motivation</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1 Unreal Development Kit Version 4 (UDK4)</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Animation Rigging Tool set (ART)</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Autodesk Maya</td>
<td>4</td>
</tr>
<tr>
<td><strong>3 Animation Design and Overview</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>4 Animation</strong></td>
<td>6</td>
</tr>
<tr>
<td>4.1 Setting up the Animation and Riggings Toolset (ART)</td>
<td>6</td>
</tr>
<tr>
<td>4.2 Why ART?</td>
<td>6</td>
</tr>
<tr>
<td>4.3 Character and Rig Setup</td>
<td>7</td>
</tr>
<tr>
<td>4.4 Animation Process</td>
<td>7</td>
</tr>
<tr>
<td>4.4.1 Exporting Animations to UDK</td>
<td>10</td>
</tr>
<tr>
<td>4.4.2 Blueprint System</td>
<td>11</td>
</tr>
<tr>
<td>4.4.3 Setting Up</td>
<td>12</td>
</tr>
<tr>
<td>4.4.4 Fixing the rig</td>
<td>13</td>
</tr>
<tr>
<td>4.4.5 Inputs</td>
<td>14</td>
</tr>
<tr>
<td>4.4.6 Blend Spaces</td>
<td>15</td>
</tr>
<tr>
<td>4.4.7 State Machine</td>
<td>16</td>
</tr>
<tr>
<td>4.4.8 Blueprint Core</td>
<td>17</td>
</tr>
<tr>
<td>4.5 AnimMontage</td>
<td>18</td>
</tr>
<tr>
<td>4.5.1 Setting Up Montages</td>
<td>19</td>
</tr>
<tr>
<td>4.5.2 Calling Montages in the Blueprint</td>
<td>20</td>
</tr>
<tr>
<td>4.6 Problems and Solutions</td>
<td>21</td>
</tr>
<tr>
<td>4.7 Animation References</td>
<td>23</td>
</tr>
<tr>
<td><strong>5 Related Work</strong></td>
<td>29</td>
</tr>
<tr>
<td><strong>6 Realization of the Animation</strong></td>
<td>32</td>
</tr>
<tr>
<td><strong>7 Conclusion</strong></td>
<td>34</td>
</tr>
</tbody>
</table>
Table of Contents (Continued)

<table>
<thead>
<tr>
<th>Bibliography</th>
<th>35</th>
</tr>
</thead>
</table>

### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>A Character Pose in Maya. This figure shows the character and rig created with the “Character Rig Creator”. It also shows, on the right, the Animations and Rigging Toolkit window.</td>
<td>8</td>
</tr>
<tr>
<td>4.2</td>
<td>The ART Interface. This figure shows a close up view of the Animations and Riggings Toolkit. Each box on the humanoid figure is a selectable joint that can be translated and rotated for animation purposes.</td>
<td>9</td>
</tr>
<tr>
<td>4.3</td>
<td>List View. This figure shows the List View tab in the Animations and Rigging Toolkit. Here you can use the S button to select all controls in that category and the V button hides all the controls in viewport in the respective category.</td>
<td>10</td>
</tr>
<tr>
<td>4.4</td>
<td>Rig Settings. This part of the Animations and Rigging Toolkit allows the user to toggle the IK/FK mode, control the Bias (squash and stretch portion of the rig) and Twist controls (for arms and legs) among other things.</td>
<td>11</td>
</tr>
<tr>
<td>4.5</td>
<td>Exporting from Maya. This figure shows the exporting window made available from the Animation and Riggings Toolkit. Here you can select what frames to export, what speed (30 frames per second, 60fps etc) and name the .fbx file.</td>
<td>12</td>
</tr>
<tr>
<td>4.6</td>
<td>Blueprints. This figure highlights the Blueprints created in your character folder as well as the character mesh and its corresponding skeleton.</td>
<td>13</td>
</tr>
<tr>
<td>4.7</td>
<td>Fixing the Rig. This figure shows that the spine in the model is currently set to follow the skeleton. This can be changed to follow the exact animation placement.</td>
<td>14</td>
</tr>
<tr>
<td>4.8</td>
<td>Before Retarget. This figure shows the character model head out of place due to the head joint being set to follow the animation as opposed to the skeleton.</td>
<td>15</td>
</tr>
<tr>
<td>4.9</td>
<td>After Retarget. This figure shows the same model joint set to follow the skeleton instead of the animation.</td>
<td>16</td>
</tr>
<tr>
<td>4.10</td>
<td>Button Creation. This figure illustrates the names of the action events created and the buttons mapped to activate/call them.</td>
<td>17</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4.11</td>
<td>Blend Space. This figure demonstrates what is shown inside of a one-dimensional blend space. A variable called speed was created and the idle, walk and run animations were placed at different intervals on the graph (bottom-middle-left). The value for speed, determined by the tilt of the analog stick, controls what animation is to be played on this graph in the game.</td>
<td>18</td>
</tr>
<tr>
<td>4.12</td>
<td>Locomotion State Machine. This is the inner blueprint setup for the Locomotion state machine. Here the game does checks to see if the character is in the air. If it is in the air it goes in the loop (JumpStart to JumpLoop to JumpEnd). If it is not in the air it simply plays the Idle_Walk_Run node.</td>
<td>19</td>
</tr>
<tr>
<td>4.13</td>
<td>Blueprint “If” statement. This figure highlights the “if” statement to go from Idle_Walk_Run to JumpStart.</td>
<td>20</td>
</tr>
<tr>
<td>4.14</td>
<td>Inside the “If” Statement node. Inside the node it shows what is required to enter the JumpStart animation. If the character is in the air it is allowed to enter that transition.</td>
<td>21</td>
</tr>
<tr>
<td>4.15</td>
<td>Inside the Jump Start node. This figure shows the JumpStart animation hooked up to the Final Animation Pose node.</td>
<td>22</td>
</tr>
<tr>
<td>4.16</td>
<td>JumpStart to JumpLoop transition. In this figure the nodes show that if 20% of the JumpStart animation is left to be played start playing the JumpLoop animation.</td>
<td>23</td>
</tr>
<tr>
<td>4.17</td>
<td>JumpStart to JumpLoop location. Shows the node that contains the logic for what was shown in Figure 4.16.</td>
<td>24</td>
</tr>
<tr>
<td>4.18</td>
<td>AnimGraph. This figure shows how UDK knows what animation to play. The State Machine named &quot;Locomotion&quot; contains the logic for transitioning between running/walking and jumping. The Slots named &quot;GroundSlashes&quot;, &quot;Fdash&quot;, and &quot;AirSlashes&quot; interact with the AnimMontage to play those animations. This node system is always running and making checks during the game. Depending on that button the player presses the appropriate animation is run through the Final Animation Pose node.</td>
<td>25</td>
</tr>
<tr>
<td>4.19</td>
<td>Blueprint Core. This figure shows the setup created to update the value of the &quot;speed&quot; variable as well as the check for if the character is currently in the air or on the ground.</td>
<td>25</td>
</tr>
<tr>
<td>4.20</td>
<td>AnimMontage creation. This figure shows how to create a new Animation Montage. Right click the folder you need to place you AnimMontage, go to New Asset⇒Animation⇒Animation Montage.</td>
<td>26</td>
</tr>
</tbody>
</table>
4.21 The Animation Montage node. The inside of the animation node. All
imported animations are located on the tab on the right. In the
middle is a screen displaying what the current animation looks
like. Below this preview is the montage segment. You drag you
animations here. Place them in any order needed, alter from what
time it starts and stops playing, to how fast the animation plays.
Right next to this is a slot named "GroundSlashes". This is how
the animations will be accessed by UDK to update into the Final
Animation Pose. On the left you can add effects and/or change the
skeleton joint locations. Below this tab is the AnimAssetDetails
tab. This tab allows you to change the speed of the highlighted
animation, how many times it loops etc. It also allows you to
enable root motion translation and root motion rotation. This tells
the game to take account for where the character is at the end of an
animation and keep the character in that new location/orientation
instead of snapping to the original position after the animation is
completed. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

4.22 Calling your AnimMontage. This figure illustrates the node setup nec-
essary to play your animation montage in game. This is connected
to the walk/run/jump setup from earlier. This shows that the char-
acter checks if the character is attacking via the "IsSlashing?"
boolean check. If an attack happens it plays the animation mon-
tage set for "Slashes". The DoOnce node makes sure it doesn’t
become stuck in an infinite loop. . . . . . . . . . . . . . . . . . . . . . . . 28

4.23 Button Solution. This figure illustrates one method of having multi-
ple animation montage setups for different buttons in the game,
simultaneously. White line connectors can only connect to a sin-
gle node, so a switch was needed. The "SwitchNumber" node is
set to a new integer value every time a button press for one of
the montages fails. This allows the blueprint to loop through each
animation montage button setup to do its true/false checks. . . . . . . . . . . 28

5.1 An Overhead Sword Attack Sequence from Devil May Cry 3 . . . . . . . . . . 30
5.2 A Slash Attack Sequence from Devil May Cry 3 . . . . . . . . . . . . . . . . . 31
5.3 A Thrust Attack Sequence from Devil May Cry 3 . . . . . . . . . . . . . . . . 31
5.4 A Slash Attack from Devil May Cry 3 . . . . . . . . . . . . . . . . . . . . . . 31

6.1 In Air animation. This figure illustrates the demo character about to
land on the floor after making a jump. It is previewed in UDK. . . . . . . . . . 32
6.2 In game Horizontal Slash. This figure shows the character doing a
horizontal outward sword swing in the game demo. . . . . . . . . . . . . . . . . . 33
### List of Figures (Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Overhead Slash. This figure shows the character at the end of an overhead slash attack in the game demo. It is similar to Figure 5.1, a screenshot of the Devil May Cry main character performing a similar overhead slash.</td>
<td>33</td>
</tr>
<tr>
<td>6.4</td>
<td>Flip Landing. This figure shows the characters landing pose after performing a front flip and coming down from the air with a vertical slash.</td>
<td>33</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction and Motivation

The video game industry is one of the fastest growing industries in the United States and gaming has become one of the most popular forms of entertainment [11]. One video game genre that has endured from the beginning of gaming is the hack and slash genre, which has evolved and reinvented itself as technologies have changed and advanced. The hack and slash style of combat is characterized by hand-to-hand combat or combat employing swords, but not guns [6] and the genre has its origins in early role playing games such as Dungeons and Dragons, where hack and slash refers to campaigns of violence with no other plot element or significant goal [3; 6].

The hack and slash genre influenced a wide range of action role playing games including Ninja Gaiden, Diablo, Chaos Legion, God of War, and the Devil May Cry series [1; 2; 5; 6; 9; 12], as well as recent games such as the Elder Scrolls series and Middle-earth: Shadow of Mordor [8; 10]. In these games, the user controls a powerful protagonist, earns or upgrades abilities for combat, and by playing well can defeat swarms of enemies. This genre is characterized by fast paced combat and usually requires memorization of enemy attacks and acute hand-eye coordination and reflexes to be successful.

This paper describes our approach to incorporating the hack and slash style of gameplay into the Unreal Game Engine (UDK4.0), a popular first person shooter video game engine. We describe our technique for creating animations using the Maya modeling tool and for incorporating these animations and actions into the UDK [7; 13]. A hack and slash game focuses on close quarters combat, primarily with weapons such as swords as opposed to guns, and as a result it boasts some of the most unique and robust character animations found in most video game genres. The animations and gameplay that we have developed are patterned after those used in the Devil May Cry series of video games.

In the next chapter we provide background describing the terms and concepts that we
use in this paper, and in Chapter 4 we provide details describing our approach. In Chapter 5 we describe games that use the hack and slash game style, with particular emphasis on the Devil May Cry series, since it has had the most influence on our work. In Chapter 6 we provide some screen captures that illustrate some of the gameplay in the video game that we have constructed, but a full appreciation of the action can only be garnered by actually playing the game. Finally, in Chapter 7 we draw some conclusions and describe future work that we have planned to develop a full fledged video game.
Chapter 2

Background

Character animation is the art of bringing characters to life. An animator uses the tools available to them, either a pencil or a computer mouse, to create characters or environments that portrays personality and emotion. Character animators turn a blank piece of paper into a believable creature with motives, thoughts and desires. An animator breathes life into still images.

2.1 Unreal Development Kit Version. 4 (UDK4)

The Unreal Development Kit, developed by Epic Games, is a powerful computer application that permits users to create a videogame, typically without requiring other game components except for model development [4]. It has its own physics engine that can be manipulated for creating your own gaming world, a plethora of effects are readily available and it boasts a new visual coding system to help out designers that understand coding logic but are not strong with syntax. Apart from all the built-in tools available, UDK4 allows the developer to import their own effects and models. It is one of the most frequently used gaming engines in game development today because of how powerful and flexible the software is. It can be used to completely create the desired project or to compliment the project.

2.2 Animation Rigging Tool set (ART)

The Animation and Rigging Tool set is a plugin created for Maya by Epic Games. The developers of UDK4 has a particular structure for how its game engine needs to view character skeletons (rigs). This plugin allows for ease of creating characters with this rig structure. It also allows animators to have a proxy character ready to animate within minutes of installing the plugin. This plugin gives you all the tools necessary for character mesh and
skeleton creation, animation process, and exporting your animations properly for use in UDK4.

2.3 Autodesk Maya

Maya, currently owned by Autodesk, is a very powerful 3D modeling software [7]. It allows for the creation of 3D polygon models, texturing, lighting, rendering, particle effects and animating. It is widely used in the film and videogame industry. Maya recognizes many different file types, such as, .fbx format (Filmbox), but mainly saves its files in the Maya Binary (.mb) format. Aside from Maya being able to do almost everything in the development of 3D modeling and animation, the fact that it can export to almost all the different 3D software file types, and the numerous plugins available for Maya, makes it one of the most robust and powerful 3D development software available. Without Maya and the ART plugin for Maya, made available by Epic Games, creating character animations would have been much more difficult.
Chapter 3

Animation Design and Overview

This project draws inspiration from a popular series of video games, the Devil May Cry series, which features a *hack and slash* style of game play [2]. Unlike the hero in Devil May Cry, Dante, the protagonist in our video game is human, although he is enhanced. He is an elite soldier in a world with a brutal history. The story line in our game begins with a huge war between magic users and technology users. This war caused magic users to become almost extinct. The fear of the awesome destructive power of magic caused it to be outlawed by the technology proficient victors after they established the new world order. Elite soldiers were enhanced through technology and tasked with keeping order and peace.

Our protagonist starts off as an elite soldier, whose’ main missions revolve around hunting down rebel magic users. Eventually magic abilities that were dormant in the protagonist awaken, which causes the people who he once called allies to hunt him down, as he did to so many others himself. He eventually learns of his true heritage and joins the rebels and helps them to overthrow the current regime. His extraordinary skill as a solider and his knowledge of technological weapons, coupled with his ever increasing magical abilities, eventually make him public enemy number one.

The animations in the game demo that we will build are reminiscent of those that appeared in the many fighting styles portrayed by the main protagonist, Dante, in the Devil May Cry series. Our character uses a magical sword that he materializes and small jet boosters for increased mobility. Due to his incredible skill with a sword, and his jet boosters, his attacks were animated to be unrealistic and explosive, yet still seem believable. Dante’s fighting style was a major motivation for the popularity of the Devil May Cry series and because Dante’s animators were able to achieve this difficult result. This, our goal, with respect to animation, is to create a character that fights with a lot of flair, is stylish, and is very technical.
Chapter 4

Animation

In this section, we describe our approach to initializing the Animation Toolset in Maya, how we use the toolset to create animations, and we provide details describing our approach to incorporating the animations into the Unreal Game Engine, UDK4. In the next section we describe Maya’s animation and rigging toolset, ART, and in Section 4.2 we provide motivation for our toolset decisions. In Section 4.3 we review the character and rig setup that we used in Maya, and in Section 4.4 we describe our use of Maya to build animations. In Section 4.5 we describe our approach to incorporating animations in the UDK and, finally, in Section 4.6 we describe some of the problems we encountered and describe our solutions to these problems.

4.1 Setting up the Animation and Riggings Toolset (ART)

The actions required to initialize the animation toolset in Maya are straightforward and can be achieved by placing a file, UserSetup.py, into the Maya scripts directory. The UserSetup.py file is located in

```
Engine\Extras\Maya\_AnimationRiggingTools\MayaTools
```

Once the file is placed in the Maya scripts directory, Maya will provide an "Epic Games" menu at the top of the Maya menu bar.

4.2 Why ART?

Typical approaches to creating 3D models in Maya prove problematic when the exported data is used in the UDK4. These problems arise because models change size during different animations. Furthermore, additional problems may arise when the root node in the rig is not set as the parent node. Character Rig Creator, which can be found in the Animation
and Riggings Toolset, solves these problems. The rig structure is streamlined for UDK4 automatically, and therefore not many problems arise when exporting files from ART to UDK4. ART is also useful for users with limited rigging experience and for those who are not accustomed to constructing a rig to be used in animations.

4.3 Character and Rig Setup

The "Epic Games" menu now available in Maya has an option called "Character Rig Creator". This is a user friendly tool that creates a basic humanoid character model that is fully rigged. For animating in this project the basic character model and rig was created. Every setting in the flowchart process of creating a rigged model using this tool was left at the default. First this tool prompts you to create a skeleton. If needed you can further define the joint setup for each limb by either adding more joints or removing joints. Skeleton Placement builds the joint mover which allows for quick placement of joints in the mesh provided. A proxy mesh is created as opposed to the usual Maya joints in this section. This can be used for animation as is and those animations can be imported into UDK on a proper character model that possesses a similar skeleton rig structure. Rig pose lets you choose the default pose for the animators to use. Usually this is set to a T-pose. Next comes the mesh weighter. This will be of use in a more refined character model to help keep the mesh smooth as the limbs contract and expand the "skin" as joints bend. Create Control Rig button opens the "Publish Character" window. Here you select a thumbnail image for your character and start the automated building of the character rig. A prompt appears after this process is complete it informs you that your character is ready to be animated and gives you the directory location of your character file.

4.4 Animation Process

Assume that the character file that was created to accomplish an animation is named ARK. It is saved as a Maya binary file (.mb). In the discussion that follows we will refer to this
Figure 4.1: A Character Pose in Maya. This figure shows the character and rig created with the “Character Rig Creator”. It also shows, on the right, the Animations and Rigging Toolkit window.

file as ARK. Upon completing the ARK file the developer is ready to begin constructing an animation. The ARK file is important so the Maya file should be backed up under a different name. For example, MainChar.mb might be created from the original ARK.mb. For new animations load the MainChar.mb file. When Maya asks for the reference file load the ARK file. The developer should rename this new file to identify its contents; for example, the file might be called Walk.mb if it’s a walk animation. Whenever a file is opened in Maya, a prompt for the reference file location appears.

The next step in the animation process is to go to the assigned to your character. Find the directory where ARK.mb is located and select that file. The developer is required to do this whenever the character file for animating is to be loaded. The rig is fairly straightforward, Figure 4.1 shows the character rig in Maya, along with the ART interface. The Animation Interface window is shown in Figure 4.2. Currently the Picker tab in the Animation Interface is displayed. This is the first of three tabs. The three tabs are Picker, List View and Rig Settings. Note the Animation Interface is needed to animate the knees of the character (where the knee is pointing). This is referred to as the Leg Pole Vector Angle controls. There is no other way to control them. Every other joint is selectable on the character in the viewport and the Animation Interface. In the animation interface click the List View tab, shown in Figure 4.3 to view the rig and select controllers similar to the Outliner in Maya.
Figure 4.2: The ART Interface. This figure shows a close up view of the Animations and Riggings Toolkit. Each box on the humanoid figure is a selectable joint that can be translated and rotated for animation purposes.

However, you cannot Right-Click anything in here to switch spaces or toggle IK/FK, for that you will need to go into Rig Settings or back into the Picker. The List View has an S and a V button at the far right of each group of controls. The S button will select all controls in that category. The V button will hide all of the controls in the viewport in that category. The Rig Settings, shown in Figure 4.4, allows you to toggle IK/FK Mode, FK Arm Orientation Space, Stretchy IK, Bias ("squash" of the squash and stretch portion of the rig) and Twist controls (used for legs and arms). Head Orientation Space will lock the head’s orientation to a body part or the world. Neck Orientation Space will lock the neck’s orientation to the chest, body, or world. Auto Hips as you animate the legs the hips will automatically react.
Figure 4.3: List View. This figure shows the List View tab in the Animations and Rigging Toolkit. Here you can use the S button to select all controls in that category and the V button hides all the controls in viewport in the respective category.

4.4.1 Exporting Animations to UDK

ART allows for exporting animations and importing animations (from motion capture data). No motion capture was used in this project. Export FBX, shown in Figure 4.5, is used to export your animation to the Unreal Editor. It will create a duplicate skeleton, and bake all of the data down onto those joints, leaving your current scene untouched. The user can choose whose motion you would like to export with the character drop down menu. If you want to export the entire animation give it a path and name and the frame range. If you would like to export your animation in multiple chunks, left-click Add Sequence to add more sequences to export. This is useful for exporting an animation that has a start, loop, and an end. To remove a sequence, Right-Click on the sequence and select remove. Export Animation will export out your character’s animation curves for all of that character’s rig
controls. Give your animation a name, and select a category to export to. The user can add new categories by left-clicking on the "+" button on the right.

### 4.4.2 Blueprint System

The Unreal Development Kit, version 4, allows developers to create games using C++, C#, Javascript and other various languages, or the Blueprint system can be used. The Blueprint Visual Scripting system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor. This system is extremely flexible and powerful as it provides the ability for designers to use virtually the full range of concepts and tools generally only available to programmers. With this system you can setup game rules and logic, cameras, player
controls, characters and character animations, items, and environments.

4.4.3 Setting Up

In a new project import the rigged character model to be used. The Figure 4.6 shows the `content browser` along with highlighted nodes. These highlighted nodes are what needs to be created in order to manipulate animations in UDK4. Initially, in the `content browser`, right click the `Game` folder to create a new folder. Name this `Character`. Then right click the character folder and create another folder. Name this `Animations`. Import the character model FBX file but dragging that file from its directory into the `Character` folder that was just created. This will create an asset for your character model and another for its skeleton (rig). In the animations folder, place all your animation FBX files from its directory into that folder. A prompt will ask you for a skeleton to use. Choose the skeleton assigned to your character.
4.4.4 Fixing the rig

Figure 4.7 shows changes that are made within UDK4 to correct your skeleton’s joint placement. A scenario may occur where the animation you imported was made on a different character model but similar rig. The ‘Skeleton Tree’ section fixes problems that may arise. In Figure 4.9, importing the character for the “Idle” animation causes the head to be sunk really low into the model. Looking at the “Translation Retargeting; tab shows that the head is set to follow the Animation. Changing the head to follow ‘Skeleton’ instead of ‘Animation’ fixes this problem. Figure 4.8 is the result of this change. This fix is universal, meaning for all other imported animations the head will be set to follow the skeleton. It should also be
Figure 4.7: Fixing the Rig. This figure shows that the spine in the model is currently set to follow the skeleton. This can be changed to follow the exact animation placement.

Note that for each individual animation you can manually translate or rotate each joint in UDK. This saves time for small fixes by eliminating the need to change the animation in Maya, export the file again then import the file another time in UDK.

4.4.5 Inputs

Figure 4.10 illustrates the 'Project Settings' and under this heading there is an 'Input' tab. This tab provides tools to create a means for the player to control the character in the game. Clicking '+' opens an extra slot in each corresponding section. Under 'Action Mappings' you create new 'Action Event' slots. Under 'Action Event' slots, the ones shown here are named 'Slash', 'Jump', 'Dash', and 'AirSlash', you create a button assignment. Shown in this figure, pressing the 'Space Bar' or the 'Gamepad Face Button Bottom' calls for the 'Jump' Action Event.
Figure 4.8: Before Retarget. This figure shows the character model head out of place due to the head joint being set to follow the animation as opposed to the skeleton.

4.4.6 Blend Spaces

In most videogames the player typically has 3 basic movements using only the movement key/keys, in this case the controller analog stick. These movements are:

1. Idle - The animation used when the analog stick is in the rest position
2. Walk - Character walks when the analog stick is moved slightly
3. Run - Character moves at max speed and runs when the analog stick is moved to its furthest position.

The Blend Space node, shown in Figure 4.11, is used to manipulate these 3 possible animations. This Blend Space that was created is a 1 dimensional Blend Space named 'Idle_Walk_Run’. We label this dimension ‘Speed’. The value for speed changes depending on how far the analog stick is tilted. From the asset browser on the right drag the needed
Figure 4.9: After Retarget. This figure shows the same model joint set to follow the skeleton instead of the animation.

animations into the 'Idle_Walk_Run' tab. At the 0 value of the speed axis place your 'Idle' animation. At the max value the 'Run' animation, and in the middle the 'Walk' animation. UDK will smoothly transition between these 3 animations depending on the value for 'Speed'.

4.4.7 State Machine

Figure 4.18 shows how the Blueprint presents the animation that is to be played. These nodes are always running during the game and checks the State Machine ('Locomotion') and the Animation Slots ('FDash', 'GroundSlashes' and 'AirSlashes') and updates the 'Final Animation Pose' accordingly. The Animation Slots access the Animation Montages that you create. This will be explained in section 4.5. A State Machine was created and named 'Locomotion'. Figure 4.12 shows the nodes needed inside of this State Machine so transition between running and jumping. The double arrow key highlighted in Figure 4.13
Figure 4.10: Button Creation. This figure illustrates the names of the action events created and the buttons mapped to activate/call them.

is a check for UDK to transition to the next node. Figure 4.14 shows inside this key. Red line nodes, as shown here, are boolean variables. This check sees if the character is in the air. If the character is in the air then it plays the 'JumpStart' animation. Figure 4.15 shows what is in the 'JumpStart' node from Figure 4.12. Simply play the 'JumpStart' animation. Figure 4.17 and 4.16 shows the check made to transition from JumpStart to JumpLoop. It shows if 20% of the JumpStart animation is left to be played then start transitioning to the JumpLoop animation; tweak that percentage for smoother transitions between animations. Similar checks are made to transition from JumpLoop to JumpEnd and from JumpEnd to Idle-Walk_Run.

4.4.8 Blueprint Core

Figure 4.19 displays the final nodes to make the character move and jump during the game. The "Event Blueprint Update Animation" node checks information on what your character is doing and feeds that information to the AnimGraph, Figure ??]. The "Is Valid?" node finds out if we are using a character. If we are not it does nothing, if we are it allows us
Figure 4.11: Blend Space. This figure demonstrates what is shown inside of a one dimensional blend space. A variable called speed was created and the idle, walk and run animations were placed at different intervals on the graph (bottom-middle-left). The value for speed, determined by the tilt of the analog stick, controls what animation is to be played on this graph in the game.

"Try Get Pawn Owner" node contains all the data for the current character. It knows if the character is moving, flying, falling etc. We use this node to extract information. "Get Movement Component" is used to help determine, in this case, if the character is in the air. "Is in air" node gets set by this node. Note, UDK is smart enough to know if a character is in the air, the "Is in air" node is a built in setter/getter for this. To set the value for speed we take the magnitude or length from the velocity.

### 4.5 AnimMontage

The AnimMontage is a very powerful tool available to animators in UDK. With this tool you can set numerous animations to play in a sequence depending on either how many times a button was pressed or how long was the button held down for or if a certain event happen, such as reloading a gun after the player runs out of ammo. You can have one animation and smartly loop that animation as many times as needed using the AnimMontage. It is also
Figure 4.12: Locomotion State Machine. This is the inner blueprint setup for the Locomotion state machine. Here the game does checks to see if the character is in the air. If it is in the air it goes in the loop (JumpStart to JumpLoop to JumpEnd). If it is not in the air it simply plays the Idle_Walk_Run node.

used to animate effects within UDK. The user can go to certain keyframes of an animation imported into a montage and select a joint where an effect can be played. For example, say your animation has the characters’ foot hitting the ground at frame seven. At frame seven you can have a dust particle effect emit from that point of impact. The AnimMontage, or Montage, was used in this project to have the main character do different sword attacks, at varying speeds, in sequence, at the push of one button. Without the AnimMontage the project would not be possible.

4.5.1 Setting Up Montages

Figure 4.20 highlights how to create an AnimMontage within you Animations folder. Several Montages were created in this project. Figure 4.21 shows an AnimMontage. This was named "GroundAttacks". The Asset Browser (right) contains all imported animations. The slot in the middle was named "GroundSlashes". Back in Figure ?? there is also a slot called "GroundSlashes" connected in that graph to the "Final Animation Pose". That slot gets the
Figure 4.13: Blueprint “If” statement. This figure highlights the “if” statement to go from Idle_Walk_Run to JumpStart.

animations from this montage. Back in the "GroundAttacks" montage, drag as many animations as you need to the slot in the middle. Place them in the correct sequence for your animations.

The user can also alter how much of the animation is played in this slot. The animations shown here are "groundAttk1", "groundAttk2" and "groundAttkFlip". These animations have a forward translation. If the animation is played in UDK during the game the character will move forward then 'snap' back to its original position at the end of the animation. To prevent this snap back and have the character stay in place you need to enable "Root Motion" found in the "Root Motion" tab located on the bottom left.

4.5.2 Calling Montages in the Blueprint

Figure 4.22 shows, in your Event Graph, the nodes needed for the player to use the Montages during the game. This setup is for the Dash Montage. Dash, and all other Montages, are setup the same way as the GroundAttacks Montage shown in Section 4.5.1 The node 'Cast To Mycharacter' links the animations to the 'MyCharacter' Blueprint shown in Figure
4.14 The Custom Event node named 'Dash' can be created after setting up a button named 'Dash' (also done earlier). This node is connected to the 'Montage Play' node which is set to play the 'DashMontage'. With this setup whenever the user presses the 'Dash' button the game looks for that Event and plays the animations found in the 'Dash' Montage. All other nodes in Figure 4.22 are checks that need to be met to perform a 'Dash' and to make sure that the 'Dash' Montage only plays once. The 'Set' node for 'SwitchNumber' will be explained in section 4.6.

4.6 Problems and Solutions

Animating the rotation on the root or master axis causes irregular behavior when exported to UDK4. The character's rotates randomly and becomes "stuck" sideways or upside down instead of returning to its default orientation after playing the animation. The rotation values in the root and master nodes must remain at its default setting of 0. Changes to this causes unpredicted behavior in UDK4 as mentioned earlier. Translating (moving in the x, y or z plane) a character in Maya from the default position for an animation and failing to translate the root node causes undesirable results. When an
animation is played in UDK4 at the end of that animation the game engine checks for the location of the root node. It then places the character in that location. Therefore, moving a character but failing to move the root node causes the character to move in the direction in UDK4 and "snap" back to its original position at the end of the animation. Animating the root node to follow the character throughout the animation results in the character moving in UDK4 and staying in its new location at the end of the animation. A big problem using the Animation Montage system is that it only work for 1 button. The Figure 4.23 is a solution for this. All white line connections from a node in the Blueprint can only be connected to 1 other node. Other colors allows a singular node to be connected to multiple nodes. Having multiple montages for multiple buttons work is impossible without a switch. The function "Switch on Int" shown in the diagram allows the white line connection, from a single node, able to connect to multiple nodes. Which node it chooses depends on the integer value the "switch on Int" function receives. Initially the node named 'SwitchNumber' is set to 0. The program checks if the button that the "0 white line" leads to is pressed, if not the value updates to 1. If false at the next check 'SwitchNumber' is set to 2. At the last button check, if false, the integer value goes back to the default 0. This is repeated until the game is shut
Figure 4.16: JumpStart to JumpLoop transition. In this figure the nodes show that if 20% of the JumpStart animation is left to be played start playing the JumpLoop animation.

down. This allows a true or false check for all buttons assigned. If any Branch check is true then that Montage animation is played.

4.7 Animation References

The goal is to create an action adventure game. Even though the game is set in the future the main character is a magic user and does not sure futuristic weapons, such as guns. He uses some technology in the form of boosters on his legs and hips. These boosters help him to dash in order to evade enemy attacks. His standard way of attacking is using magic to summon swords. The swords materialize, he grabs it, does an attack then the sword dissipates at the end of his attack. The game is heavily influenced by the games Devil May Cry 1 and Devil May Cry 3. Those games are considered hack and slash games. They consist of medium to easy, in difficulty, puzzles but primarily focus on defeating hordes of enemies with a wide variety of moves and strategies. In those games you play a very powerful demon hunter named Dante, who is part demon himself and part human. He uses enchanted swords and guns to destroy his foes. The combat is very fast, highly stylized, combo oriented and rewards players who possess quick reaction times to deal
Figure 4.17: JumpStart to JumpLoop location. Shows the node that contains the logic for what was shown in Figure 4.16

with enemy attacks. This project tries to capture the core essence of those games while building something unique of our own. Videos of various martial artists using swords and fictional sword styles, from movies and games, were used as reference to help create the animations for the main characters’ fighting style. It is an effort to create a very unrealistic and stylized combat maneuvers yet somehow make it believable. That is the goal this projects’ animations.
Figure 4.18: AnimGraph. This figure shows how UDK knows what animation to play. The State Machine named "Locomotion" contains the logic for transitioning between running/walking and jumping. The Slots named "GroundSlashes", "Fdash", and "AirSlashes" interact with the AnimMontage to play those animations. This node system is always running and making checks during the game. Depending on that button the player presses the appropriate animation is run through the Final Animation Pose node.

Figure 4.19: Blueprint Core. This figure shows the setup created to update the value of the "speed" variable as well as the check for if the character is currently in the air or on the ground.
Figure 4.20: AnimMontage creation. This figure shows how to create a new Animation Montage. Right click the folder you need to place you AnimMontage, go to New Asset⇒Animation⇒Animation Montage.
Figure 4.21: The Animation Montage node. The inside of the animation node. All imported animations are located on the tab on the right. In the middle is a screen displaying what the current animation looks like. Below this preview is the montage segment. You drag your animations here. Place them in any order needed, alter from what time it starts and stops playing, to how fast the animation plays. Right next to this is a slot named "GroundSlashes". This is how the animations will be accessed by UDK to update into the Final Animation Pose. On the left you can add effects and/or change the skeleton joint locations. Below this tab is the AnimAssetDetails tab. This tab allows you to change the speed of the highlighted animation, how many times it loops etc. It also allows you to enable root motion translation and root motion rotation. This tells the game to take account for where the character is at the end of an animation and keep the character in that new location/orientation instead of snapping to the original position after the animation is completed.
Figure 4.22: Calling your AnimMontage. This figure illustrates the node setup necessary to play your animation montage in game. This is connected to the walk/run/jump setup from earlier. This shows that the character checks if the character is attacking via the "IsSlash-ing?" boolean check. If an attack happens it plays the animation montage set for "Slashes". The DoOnce node makes sure it doesn’t become stuck in an infinite loop.

Figure 4.23: Button Solution. This figure illustrates one method of having multiple animation montage setups for different buttons in the game, simultaneously. White line connectors can only connect to a single node, so a switch was needed. The "SwitchNumber" node is set to a new integer value every time a button press for one of the montages fails. This allows the blueprint to loop through each animation montage button setup to do its true/false checks.
Chapter 5

Related Work

The goal of this project is to create an action adventure game frequently referred to as a “hack and slash” game. There is a wide variety of games in this genre but the animations that we target in this thesis are inspired, for the most part, by one of the early pioneers of the genre, namely the games “Devil May Cry 1” and “Devil May Cry 3”.

*Devil May Cry 1* was released on August 23, 2001 for the Playstation 2 by the game company “Capcom.” It was the first game of its kind that allowed players, in an action adventure game, to perform unique combination moves, or *combos*; examples of combos might be juggling enemies in the air with bullets and then seamlessly switching back to close ranged melee attacks.

*Devil May Cry 1* received a lot of praise from videogame reviewers and fans. It became a cult classic that inspired numerous other titles, such as *God of War*, *Chaos Legion* and the new *Ninja Gaiden* games. The gameplay idea of the *Devil May Cry* series is to have the user control a very powerful and skilled antagonist. The user is tasked with completing numerous stages that may or may not have a boss battle at the end.

Capcom released *Devil May Cry 2* shortly after but critics and most fans considered it a failure, partly due to the high quality of the first game, and the lower quality of the followup. As a result Capcom revamped the game engine, the graphics, and the combat system, and emphasized the story in the form of cutscenes at the beginning and end of each level. The result of these improvements was a game called *Devil May Cry 3*, and they released the revamped game on February 17, 2005 for the Playstation 2.

*Devil May Cry 3* is regarded as the best entry in the *Devil May Cry* series by numerous fans and videogame critics. It has been praised for its return to being a very difficult game to beat, amazing combat system and for fleshing out *Dante*, the main protagonist and chief
character of the story.

The combat system in Devil May Cry 3 is the main influence for this projects’ combat system. New styles were introduced into Devil May Cry 3 so that users could add their own unique flavor to how they wanted to fight enemies. Ten new melee and projectile weapons were added to the game, each with unique moves and properties. This was also amplified by the new options certain styles opened. For example, the gunslinger style allows the player to unlock a projectile weapons’ full potential by giving access to almost double the amount of different moves for each gun. Swordmaster style allows access to more powerful and more unique combos with the melee weapons. These elements, along with the games’ new air combo system, created a new limitless amount of different combos a player can perform. It is so robust that there are tournaments where players would perform combos in a stage and are judged against other players’ performance in the same stage.

Figures 5.1, 5.2, 5.3, and 5.4 are screenshots of the main character performing a move in the game.

Figure 5.1: An Overhead Sword Attack Sequence from Devil May Cry 3
Figure 5.2: A Slash Attack Sequence from Devil May Cry 3

Figure 5.3: A Thrust Attack Sequence from Devil May Cry 3

Figure 5.4: A Slash Attack from Devil May Cry 3
Chapter 6
Realization of the Animation

To demonstrate the feasibility of our approach, we have implemented the techniques described in previous chapters and incorporated them into a video game constructed with the Unreal Game Engine, UDK4. We illustrate our implementation through figures, included in this chapter, which describe our implementation of character animations that achieve the same look and feel as those used in games that exemplify the *hack and slash* genre. In particular, Figures 6.1, 6.2, 6.3 and 6.4 show screen captures illustrating the actions that can be performed in the video game built with the animations described in this thesis. These animations illustrate the player character in the game attacking with a large sword. The actions are similar to those shown in the previous chapter in Figures 5.1, 5.2 and 5.4.

We also provide video capture of these actions performed while playing the video game for the project; however, full appreciation of the animations can only be achieved by playing the accompanying video game.

Figure 6.1: In Air animation. This figure illustrates the demo character about to land on the floor after making a jump. It is previewed in UDK.
Figure 6.2: In game Horizontal Slash. This figure shows the character doing a horizontal outward sword swing in the game demo.

Figure 6.3: Overhead Slash. This figure shows the character at the end of an overhead slash attack in the game demo. It is similar to Figure 5.1, a screenshot of the Devil May Cry main character performing a similar overhead slash.

Figure 6.4: Flip Landing. This figure shows the characters landing pose after performing a front flip and coming down from the air with a vertical slash.
Chapter 7

Conclusion

In this paper, we have described our approach to incorporating Devil May Cry style animations into the Unreal Game Engine, the UDK 4[13]. To build the animations we used the Maya modeling tool, together with the UDK 4 Game Engine, to realize the animations into an actual video game. The motivation for our development is the burgeoning popularity of the hack and slash genre, and our interest in investigating the feasibility of incorporating hack and slash animations into a first person shooter game engine. We believe that our animations are faithful to the spirit of the hack and slash genre and, in particular, to the fast, stylish, and challenging game play demonstrated in the Devil May Cry series.

Our future work includes further development of the animations, with emphasis on aerial combination animations. We also plan to incorporate enemy artificial intelligence, improvement on the character models, music, and sound. Our final goal is the development of a game level to post on a kick starter web page to obtain funding for a full fledged hack and slash video game.
Bibliography


