UDK "Musical" Collision Environment Experiments

Stephen Lucas December 2013

For video documentation, see http://www.youtube.com/watch?v=lKMY3_gONFs&list=PL_1souUpUDC0nE-QE8PFDRVvDMJmB4vgI

Introduction

This project is primarily an exploration of the possibilities of using Unreal Development Kit (UDK) as an interactive/generative music environment and instrument. Instrumentalists are distinguished by their ability to react with their instrument in the moment and place sounds in time. However, musical performance must also exist in space, requiring performers to also react with the properties of the performance space and adjust their technique to the specifics of the environment. In a virtual environment like in a video game, dimensions of time and space can be malleable and interchangeable, allowing for more experimental organizations of musical constructs. Likewise, the immersion of the player as both performer and audience allows for more open-ended considerations of musical narratives. A game engine such as UDK provides flexible tools for creating such an environment, but given that this is not the typical goal of the game engine, it is difficult to adapt techniques that are common in computer music systems and 3D animation. Additionally, previous techniques in this mindset have very limited documentation and the burden on a developer coming from outside game development is very high. This project seeks to document some interesting techniques that are possible with physics and sound interactions in UDK, so that future development in this direction can be advanced.

Motivation

Games are much like instruments because they provide a fixed framework for a person to interact with. An instrument is a collection of functions and limitations, shaped by a history of performance practice; there may be a cultural tendency toward an instrument's use, but the possibilities for its being acted with are not necessitated by any permanent goal or requirement beyond the bounds of physics. Likewise, a game may have a preconceived narrative or goal, but the primary function of a game is to engage the player with a construct. Although it can be enjoyable to play into the preconceived narrative of a game, when the player finds a way to break out of the planned path of the game and creates their own context for experience, the separation of game designer and game player breaks down. All art requires the viewer to use their own experiences to create the context for their viewing, but "pieces" like games allow for a level of interaction that heightens the feedback where that viewing context can be reshaped in real time.

3D Physics simulations are intriguing because they offer a high level of intuition as to how objects interact, but the mapping of interactions of several objects onto other structures and increasing the scale of dimensions quickly create a complexity of information that can become unpredictable. Allowing a user to interact with a simulation in real time reclaims some of the intuitive nature of how objects exist in the constructed space, making it possible to virtually explore the mappings.

This project is novel because objects represent sound events in time and pitch similar to a MIDI editor. However, the objects exist as physical objects and are as unfixed in space as they are in time. A first-person-shooter type interface allows a player to directly experience this disconnect. This can be conceived as a musical instrument, but there is not an imperative for creating the traditional narrative or aesthetic of a performance piece.

Related Work

Graham Gatheral's work with UDK was a heavy influence in this project. UDK offers many techniques for creating objects and simulations, but the possibilities are still quite limited. Gatheral has published several examples of exporting dynamic data from UDK through OSC to SuperCollider to drive audio synthesis.[1] Specifically, his experiments with metal impacts demonstrated that it is possible to export accurate collision information from UDK in real time.[2] This may seem trivial, but physics engines do now always have methods for exporting the collision data; sometimes this can be approximated by selective computer vision of cross sections of 3D space, but given that the collision data exists somewhere in the simulation, it is always preferable to have direct access to it.

Supercollider allows for far more extensive possibilities in sound synthesis, but it is interesting to try and stretch the possibilities of audio within UDK. Unfortunately, the most direct method is playing samples with Kismet (UDK's built in logic editor) and controlling the playback rate (affecting pitch) and volume. Other audio effects are possible in UDK, but applying these dynamically requires odd tricks of cross fading multiple instances of the same sound with varying levels of the effect.[3] This makes it more difficult to apply this technique to multiple sounds, as Kismet does not have a direct way to instance this process and it would have to be manually duplicated for each sound.

One technique that was investigated was applying variable gravity fields to the physics simulation so that the motion could be influenced by more than the player interactions. A first example

1 "UDK + Supercollider: Real-time sound synthesis for Unreal," Graham Gatheral, Accessed December 12, 2013, http://www.gatheral.co.uk/tutorials/udk-supercollider

2 "UDK + Supercollider for real-time sound effect synthesis - demo 3," Graham Gatheral, Accessed December 12, 2013, http://www.youtube.com/watch?v=Ugh_6ncZnlk

3 "Apr 22: Accessing the Continuous Modulator Node," Graham Gatheral, Accessed December 12, 2013, http://www.gatheral.co.uk/blogs/udk/index.php?/archives/15-Accessing-the-Continuous-Modulator-node.html

of this is "The Physics Dance!," which demonstrates multiple physics blocks being thrown around a room.[4] However, the documentation for gravity fields is poor and it appears that this video uses multiple fixed gravity fields as it is impossible to create these fields with dynamic position. In another example, "Egypt Alive," it is much more apparent that the relative simplicity of automating the intensity of a few static-position gravity fields is made more interesting by directing the camera and dynamic light sources to add interest.[5]

For this project, it was decided that it would be more interesting if these fields could be directly influenced by the player, so as to overcome the inability to produce dynamically moving fields. However, building additional player interactions like this is not trivial, so the manner of physics interactions had to be limited to basic functions of grabbing and throwing individual objects. Building more options for influencing gravity fields would be a good future direction for increasing the interest of the physics simulation.

An important aesthetic goal was simulating the transport line of a MIDI playback engine in 3D space. The tutorial by *darthwilson*, shows how to automate a plane with Kismet and build an interesting material "as seen in games such as Batman Arkham Asylum and Mass Effect."[6]

One of the most ambitious projects for expanding UDK into a musical environment is UDKOSC, which is more specifically oriented toward exporting data from/to UDK over OSC.[7] This project is similar to Gatheral's work, but with a more extensive framework for data transmission. Although UDKOSC incorporates particle generation and collision, they are tied to particle generators and fields and not distinctly physics objects that can be individually manipulated by the player.

Approach and Schedule

The primary work for this project is the determination of how much can be done with collisions and audio synthesis/playback in UDK. The learning curve for UDK is steep, so time must also be distributed to documentation and troubleshooting. The time was allotted in (approximately) the following order with some overlap and switching between phases as part of the work flow.

40% - Research and tutorial exploration

20% - Troubleshooting

10% - Tweaking visual aesthetics

---

4 "UDK - The Physics Dance!," Elude87, Accessed December 12, 2013, http://www.youtube.com/watch?v=W_ZDNTIJAF4

5 "Egypt Alive! - UDK Physics Music Video," Josh Caulton, Accessed December 12, 2013 http://www.youtube.com/watch?v=oBR065WkZw8

6 "UDk Scanner Tutorail," darthwilson, Accessed December 12, 2013, http://www.youtube.com/watch?v=KoX0nji7n0s

7 "UDKOSC - CCRMA Wiki," CCRMA, Accessed December 12, 2013, https://ccrma.stanford.edu/wiki/UDKOSC

10% - Implementing physics simulation

10% - Audio design and implementation

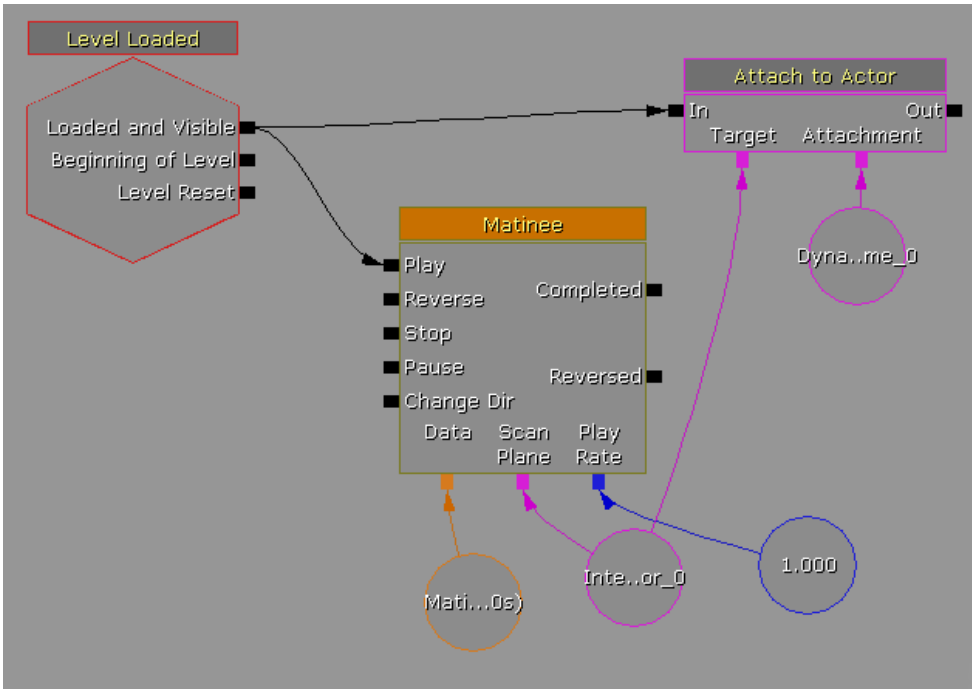
10% - Documentation

Methodology

A comprehensive explanation of all of the techniques used in UDK is not appropriate here, but a focus is on the primary problems in implementing audio and collisions is useful. It is hoped that people developing similar projects can use these directions to bypass some of the confusion from previous documentation and examples. These are not necessarily the only or best ways to accomplish these goals, but they were the most effective found. Also, much of the implementation was not accomplished with UnrealScript, which could potentially be much more effective for accomplishing the goals. Some of these explanations may seem rudimentary, but in the interest of helping beginners, they are as detailed as seems useful.

Scan Plane

The primary collision goal was to find the point of impact between a non-blocking scanning plane with dynamically positioned physics objects. The plane was imported as a static-mesh from the default content. The important element for animating triggering a Matinee with a Level Loaded event.[8]



The Matinee has a "Movement" property to animate the plane. The "Play Rate" float allows the rate of scan to be set; currently, this is just set in Kismet and is not changeable during the game. The plane must be made as an InterpActor to be animated (here "Inte..or_0"). In order to get collisions later, you

8 Based on darthwilson video tutorial.

must made an overlapping plane that is an invisible DynamicTriggerField and use the "Attach to Actor" action with the trigger field as the attachment (here "Dyna..me_0"). Annoyingly, it seems that you can only make trigger fields via the builder brush, but you would not necessarily want the trigger field to be as thin as a 2D plane.

Collisions

Getting collision positions requires you to set up individual KActors and then integrate them with Kismet. In theory, you should be able to dynamically generate the KActors rather than making them all manually. However, it was not clear how to get the dynamically generated KActors into Kismet and be able to differentiate objects that should be associated with different sounds. One attempt was to associate different sounds with render materials so that at the time of collision, a "Get Propery" action was used on the object to determine it's material. Unfortunately, there was a failure to retrieve the material string in Kismet so objects all needed to be set manually.

The three most important properties for the KActors are:

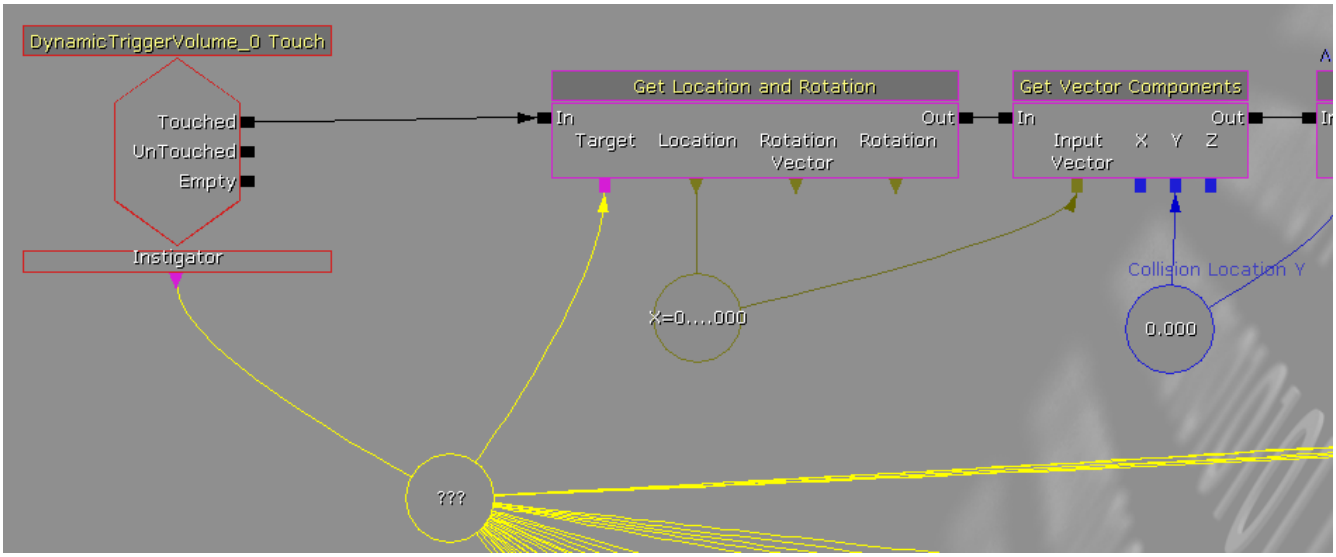
No Encroach Check = FALSE

Physics = PHYS_RigidBody

Collision Type = COLLIDE_BlockAll

RBChannel should be the same as the trigger field

In Kismet, the Touch event is monitored on the trigger volume with the Instigator becoming the KActor it touches. This is then used to determine the location vector, and from there, the Y position. Potentially Z position could also be used. In this case, X position is the position where the trigger field is, so it will already be time as it scans.

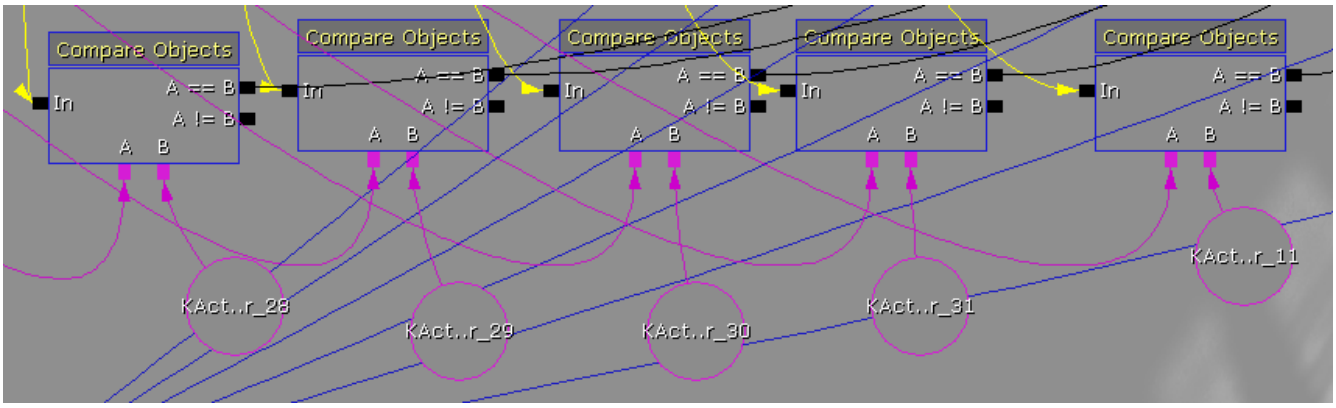


One problem with the physics is that they are only reported when the instigator is moving. This works

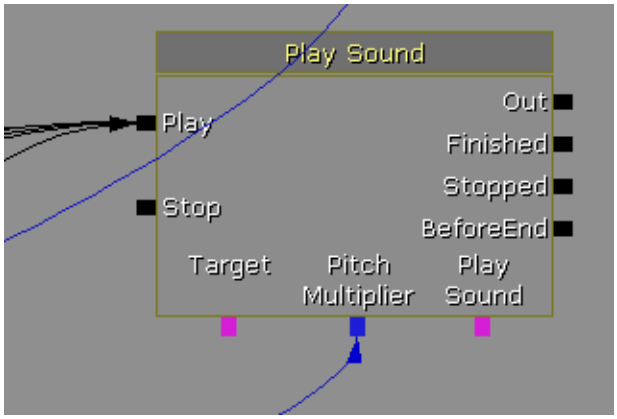
out well for the KActors being spheres because they will almost always be rolling, but this would be a problem for other shapes. One potential solution is to set up a very low intensity gravity field over the entire space so that the KActors are minutely wiggling and would trigger touch events.

Sound Triggering

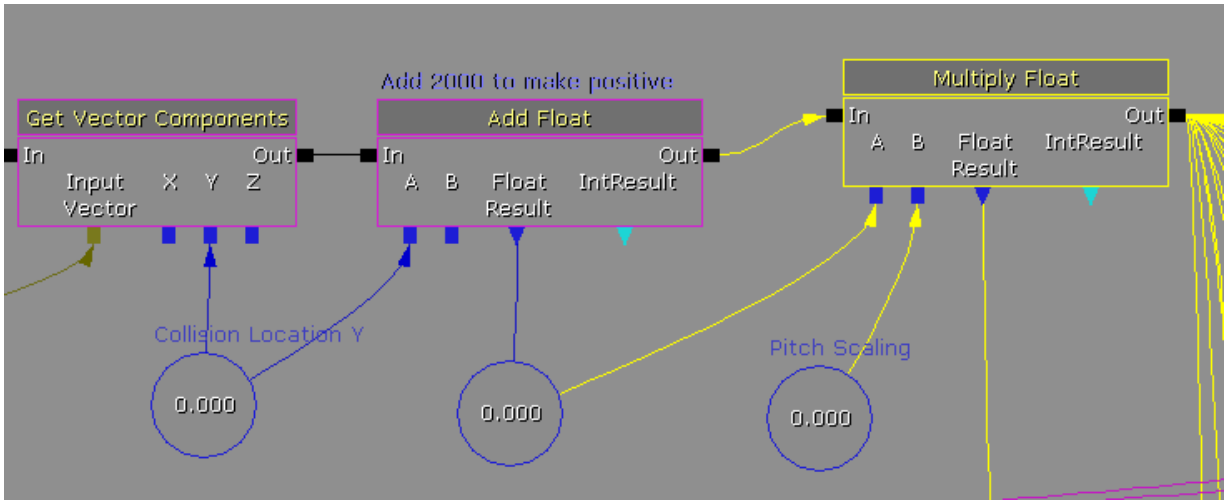
To determine which sound should be played, the Compare Objects action manually looks for each of the KActors. If the comparison between the Touch event object matches, it will trigger the corresponding sound.



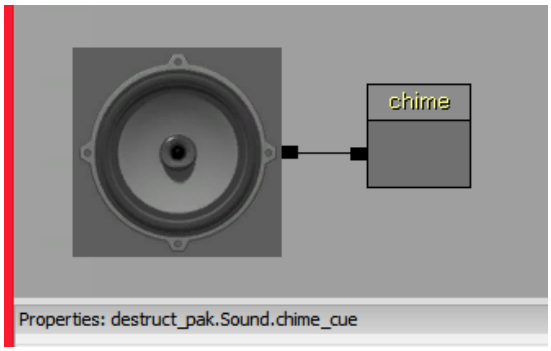
Playing the sound requires a Play Sound action with the sound cue manually set.



In this case, the Pitch Multiplier float is related to the Y position from the Touch, but it must be scaled first. In this case the values are all made positive by an offset of 2000 and are then scaled down to meet the confines of the Pitch Multiplier.



Note that "Sound Cues" are what is played and they contain the sound files as "SoundNode Waves"



Player and Weapon

Properties of the player character like it's movement and weapons must be accomplished by writing classes and changing the game type. The most important changes were made in a new class, MyPhysGameInfo.uc

class MyPhysGameInfo extends UTDeathmatch;

defaultproperties

{

PlayerControllerClass=class'MyPhysPlayerController'

DefaultPawnClass=class'MyPhysPawn'

DefaultInventory(0)=class'UTWeap_PhysicsGun'

Name="Default__MyPhysGameInfo"

}

A main change here is the MyPhysPawn.uc class which changes the movement speed and jump intensity.

class MyPhysPawn extends UTPawn;

DefaultProperties

{

JumpZ=+1000.000000

GroundSpeed=+1500.00000

MaxFallSpeed = 0.0

Name="Default__MyPhysPawn"

}

Vehicles were also included as an easy way to expand player motion and actions (weapons). None of the default sounds of the player or the vehicles were removed. Originally, it was predicted that it would be interesting to make the character sounds similar to the musical events. However, the current state of the project is not geared toward a pristine audio experience, so the starkness of the character sounds exposes the project as an exploration in documenting the techniques.

Compiling

Unfortunately, there is still a problem with the game compiling for OSX. The Physics game works but the weapon and player character properties are not correct. Additionally, configuring the compiler settings to make the game start automatically is difficult. A post on StackExchange seemed to help with this the most but it is still problematic to get the compiler to do exactly what one wants.[9]

It takes some time to setup, but afterward building new version is pretty straightforward.

First, update config file UDKGame\Config\DefaultEngine.ini or DefaultEngineUDK.ini.
There are 4 sections you need to update: [Engine.ScriptPackages], [UnrealEd.EditorEngine], [Engine.StartupPackages] and [Engine.PackagesToAlwaysCook].

[Engine.ScriptPackages]

Add here lines for all new script packages like this:

```
[Engine.ScriptPackages]
+NonNativePackages=UTGame
+NonNativePackages=UTGameContent
+NonNativePackages=NewScriptPackage
```

If you do not use classes from UTGame and UTGameContent then it`s safe to remove them from config.

[UnrealEd.EditorEngine]

You need to add there lines for script packages. That settings is used to include scripts for compiling step, so there are high chances that settings are already done.

[Engine.StartupPackages]

Add lines for content packages at this section. You need to add here only packages that need to be loaded all time, e.g script packages, font packages. Also include here packages resources from are referenced

9 "It takes some time to setup, but afterward...," the vk, Accessed on December 12, 2013, http://gamedev.stackexchange.com/questions/23514/get-final-output-from-udk

*dynamicaly via **DynamicLoadObject()** method. Otherwise such packages will not be cooked.*
[Engine.PackagesToAlwaysCook]
You need to add here packages that are not referenced from other packages but still need to be distributed, e.g. entry level map.
After this setup fire up UDK Frontend and run build cycle - recompile, cook, package. After this you will get installer with cooked game. That game only has executables, configs, maps, separate texture file and Startup.upk package with all other resources.

Documentation

Because there were still some problems with the compile, the video documentation was played in the UDK editor; this also made it simple to demonstrate the properties in the editor and Kismet. It is difficult to get clean screen capturing with processor intensive games at high resolution. The preferred method is to use a hardware recorder like the BlackMagic Hyperdeck Shuttle 2 to intercept the video going to the monitor and record. Unfortunately, this device was not directly compatible with the testing computer and was not used. *Action!* by mirillis was found to get the best quality capturing (the trial includes a watermark but it wasn't in the way on the screen capture).[10]

Evaluation Method

One important evaluation is of how good UDK is with dealing with physics collisions. Firstly, the amount of collision data reported is very good. The ability to reference the two objects in the collision so specifically and neatly in Kismet is a good, but there should still be a better method for referring to several instances of the same or similar objects by something like material or a unique property. The efficiency of the physics simulation in real time is good, even when it means there are dynamic shadows being generated. Potentially, raising the number of objects much higher would still cause a problem. In general, it is much faster to get a rendered result from UDK than in 3D animation software.

Because most of this project became about expanding the documentation of working with audio and collisions solely in UDK, the "musical" results are not as developed as originally desired. Playing the game will intrigue the player more on the level of how they interact and associate with a sandbox-style game space, rather than other musical games that have more abstract, intuitive, and/or pleasing musical results. It is likely that this project will be attempted in another game engine to see if some of the shortcomings of UDK could be overcome. However, the biggest success for this project may be that it helps clarify UDK for new users so that they don't have to encounter the same amount of difficulty in implementation.

10 "Action! - The world's No. 1 gameplay recorder and Windows desktop recorder!," mirillis, Accessed on December 12, 2013, http://mirillis.com/en/products/action.html

Bibliography

Caulton, Josh. "Egypt Alive! - UDK Physics Music Video." Accessed December 12, 2013. http://www.youtube.com/watch?v=oBR065WkZw8

CCRMA. "UDKOSC - CCRMA Wiki." Accessed December 12, 2013. https://ccrma.stanford.edu/wiki/UDKOSC

darthwilson. "UDk Scanner Tutorial." Accessed December 12, 2013. http://www.youtube.com/watch?v=KoX0nji7n0s

Elude87. "UDK - The Physics Dance!." Accessed December 12, 2013. http://www.youtube.com/watch?v=W_ZDNTIJAF4

Gatheral, Graham. "Apr 22: Accessing the Continuous Modulator Node." Accessed December 12, 2013. http://www.gatheral.co.uk/blogs/udk/index.php?/archives/15-Accessing-the-Continuous-Modulator-node.html.

Gatheral, Graham. "UDK + Supercollider for real-time sound effect synthesis - demo 3." Accessed December 12, 2013. http://www.youtube.com/watch?v=Ugh_6ncZnlk.

Gatheral, Graham. "UDK + Supercollider: Real-time sound synthesis for Unreal." Accessed December 12, 2013. http://www.gatheral.co.uk/tutorials/udk-supercollider.

mirillis. "Action! - The world's No. 1 gameplay recorder and Windows desktop recorder!." Accessed on December 12, 2013. http://mirillis.com/en/products/action.html

the vk. "It takes some time to setup, but afterward..." Accessed on December 12, 2013. http://gamedev.stackexchange.com/questions/23514/get-final-output-from-udk