Kinect Drum Mods

Nick Aversano, Alan Buabuchachart, Michael Eleff,
Sri Kankanahalli, Brendan Rowan

December 13, 2013

Abstract

We have built an intuitive, playable interface in which a user uses a Kinect to not only create and play drums, but also modify their appearance and sound. It's a unique experience that lets the user fully interact with a virtual drum within a physical space. Our goal is to make it as close to playing real drums as possible. Although the control is currently imperfect, we are planning to improve it.

1 Introduction

Graphical user interfaces (GUI) are often limited by the imagination of the designer. Users are often unable to utilize the full power of a piece of software when it is unintuitive. There is little or no chance that the designer of a program can create an interface which will conform to the train of thought of all users. This will detract from even the most powerful algorithms because they cannot be used to their full capabilities by users.

The study of human-computer interaction is a young science and while much has been done to make computers more intuitive, the interfaces between users and software are created before the software is even deployed. Being as such, there is no concern being put forth to personalize the interface to a user at runtime.

Our vision is of an interface for playing music which does not force the use of pre-drawn graphics. Current interfaces bind the user by utilizing only the programmer's imagination and his or her idea of the appearance of an instrument. We created an interface whereby the size and pitch of an instrument is decided by the user. Using gestures, a user can resize a drum and, by virtue of this manipulation, change the drum's pitch.

To accomplish our goal of creating a more dynamic interface for playing a drum, we divided into groups to control graphics and sound. Graphically, the realism of the drums comes from the user's ability to change a drum's shape in each direction as well as alter the drum's shape. Our implementation, as well as insights are discussed in this paper. The order of the contents is system design, control, graphics, sound, and finally, our evaluation and concluding thoughts.

2 Related Work

Prior research has been done on creating customizable interfaces with the Microsoft Kinect. Microsoft Research has developed BodyAvatar, a system that allows users to create free-form, customizable avatars that can be attached to the body.[1] In essence, BodyAvatar allows the user to become the avatar as opposed to the user creating an avatar in a separate environment entirely. The user is able to edit their avatar in real-time and the avatar is by default attached to an image of the users body, through the Kinect's technology. Furthermore, the user can detach certain body parts from the avatar, thus creating other extremities like a tail or hair. Because body parts can be detached and reattached in the system, other body parts can be binded to certain areas of the avatar which allows for stretchable or compactable components. The avatar can also be colored, allowing for unique views of the same shape. Users might want to color their avatar to be their favorite color or give their avatar eyes.

Similar to Microsoft Research's BodyAvatar, University of Purdue's Shape-It-Up[2] and Handy-Potter[3] take advantage of the Kinect to manipulate 3D object using hand motions. However, they took it a step further by also integrating gesture detection to allow for more complex control. They achieved this by using machine learning algorithms to classify the different gestures. As a result, they were able to create controls that are more intuitive and feature-rich than those of our current system.

Recent work has been done at Hack Duke, where Duke University students created Air Instruments, a Kinect application that allows for users to play air instruments such as drums and guitar.[4] Similar to the game Rock Band, the instruments on screen are prerendered such that the user has no customization options. However, Air Instruments successfully allowed for different instruments to be played, which could be through strumming or banging drums, at a single time.

3 Procedure

Our application is built using Windows Presentation Foundation (WPF). WPF is a graphical subsystem for rendering user interfaces in Windows-based applications. We decided to use WPF instead of a simpler system (e.g. Windows Form) because it allowed us to render 3D graphics without requiring external libraries such as OpenGL.

Our application is split into 4 main components: system design, control, graphics, and sound. System design deals with the flow of the program and how different components communicate with each other. Control deals with how the user interacts with the drums. Graphics deals with how we represent the drum on the screen. Finally, sound deals with how we output the sound based on the shape of the drum.

[1]Zhang, Yupeng, et al. "BodyAvatar: creating freeform 3D avatars using first-person body gestures." *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013.

[2]Murugappan, Sundar, HaiRong Liu, and Karthik Ramani. "Shape-It-Up: Hand gesture based creative expression of 3D shapes using intelligent generalized cylinders." *Computer-Aided Design* (2012).

[3]Murugappan, Sundar, Cecil Piya, and Karthik Ramani. "Handy-Potter: Rapid Exploration of Rotationally Symmetric Shapes Through Natural Hand Motions." *Journal of computing and information science in engineering* 13.2 (2013).

[4]Browne, et al. "Air Instruments." *Hacker League*. 17 November 2013. Web.

3.1 System Design

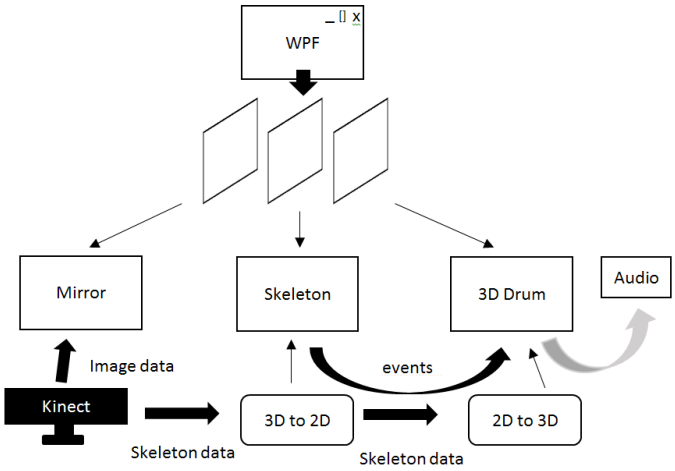

Figure 1: *Overall system design*

Figure 1 illustrates the overall design of our application. We started with a WPF window and added 3 different layers on top: the mirror layer, the skeleton layer, and the 3D drum layer. The mirror layer receives image data from the Kinect and displays it onscreen. The goal here was to make a user feel as if he/she is interacting with the drum in a physical space. The skeleton layer takes the tracking joint data, converts it from 3D coordinates into 2D coordinates, constructs a skeleton from all the joints, and displays it on the screen. The skeleton layer acts like an avatar for the users without actually replacing them (since we don't want to disengage the user from the physical space) and helps the user understand how he/she can interact with the virtual space. Once the skeleton layer finishes drawing, it sends the joint data over to the 3D drum layer. The 3D drum layer converts the data back into 3D coordinates, then performs a hit test on the joints with the drums, moving, resizing or playing them accordingly. The conversion from 3D to 2D and back to 3D were necessary because the Kinect outputs 3D data, but skeletons are drawn into a 2D plane and the drums exist in a 3D plane. The application is an event-based application, so all sending and recieving of data was done through event handling.

3.2 Control

Figure 2 shows the various hit boxes we have for each drum. By touching the top with his/her right hand and the bottom with his/her left hand, the user can resize the height of the drum. To move the drum, the user simply touches the body with his/her left hand, holds onto the drum, and moves it to a desired location. To stop moving the drum, the user can touch any location on the drum with his/her right hand to release it. The user can also independently change the diameter of the top and the bottom of the drum by using both hands to simultaneously touch TL and TR or BL and BR, then moving their hands to resize accordingly. To play the drum, users simply hit the top with their left or right hand from any location above the drum.

One of the main challenges we faced with the controls was the fact that we are limited

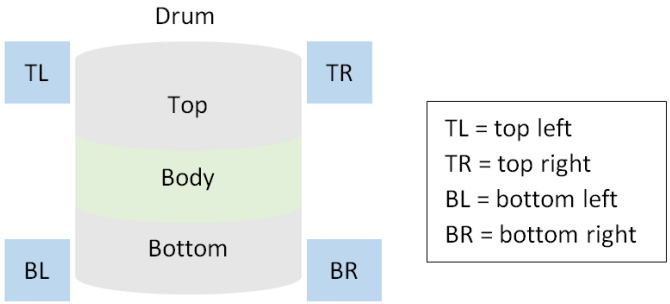

Figure 2: *Control hit boxes*

to only joint data, so we can't perform gesture detection. Without gestures, we can only use basic motions to manipulate the drum. This led to a problem of conflicting motions, where similar motions perform different tasks. For example, to move a drum, the user touches the body with his/her left hand and moves it, but the user can also play the drum with his/her left hand by touching the top. If the user's hand moves through the top to the body while trying to play the drum then he/she will start moving the drum. We tried to fix this by implementing a cooldown period where the user can't move the drum right after playing it. Although this alleviated the problem somewhat, it wasn't able to get rid of it entirely. We are still investigating why.

3.3 Graphics

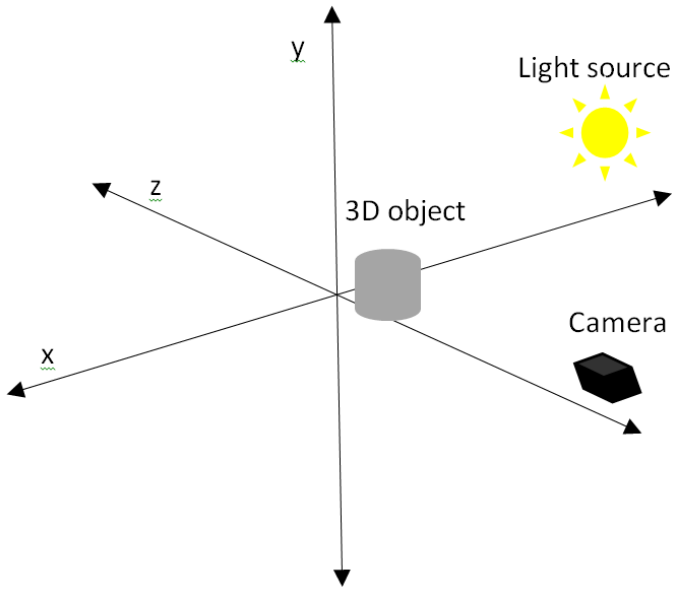

Figure 3: *Light source and camera positions within 3D plane*

Before creating the 3D version of our program, we first experimented with a 2D version where the user interacted with a simple square. Everything was easy and straightforward in the 2D world, but moving to a 3D version introduced many more challenges. We now had to

deal with the camera, light sources, mesh construction, object surface, and 3D coordinates.

2D coordinates start from the top left coordinate (0,0), only expanding down and to the right. However, 3D coordinates start from an arbitrary origin point (0,0,0) and expand in all directions. To find the proper origin, we had to set up the virtual camera correctly. If the camera points a bit to the left or a bit to the right, the user can't see anything; however, if it points dead straight into (0,0,0), the 3D world looks unnatural. Once we set up the camera correctly, the user still won't be able to see any object without a virtual light source. We also had a similar problem with the light source, where we had to set it up at a correct position. If the position is off, then the light shines strangely, and objects look too bright or too dark) The way the light reflects on an object's surface also depends on how we calculate the normal of the points on the object, so we had to get that working correctly as well.

A 3D object consists of a collection of triangles in 3D space. To construct a 3D drum, we created a set of 3D points and connected them in triangles such that the object looks like a drum; we based our 3D drum on a cylinder to help simplify the process. Once we have a cylinder, we can dynamically modify it (e.g. changing its height and radius).

3.4 Sound

The main problem within the sound portion of this project was changing the pitch of a drum based on its size. The pitch change needed to be dynamic and on the fly, with little to no delay between the user hitting the drum and sound playing, so instead of changing the actual pitch of the drum sample (which would involve fairly intensive calculations, such as a Fast Fourier Transform) we modify the sample's playback rate. This has a similar effect, but with the caveat that it also affects the length of the sample; for example, if the drum plays at twice its original pitch, the sample will also be half of its original length. For the purposes of this project, however, we found that this was acceptable.

When implementing this in practice, we first copy the original drum sample (a 44100Hz .wav file of a snare) over to a new sound stream. Then, we change the playback rate in the stream header, according to the following multiplier:

$$R = 44100 * (1 - (\frac{s - m}{M - m}))$$

where R is the new playback rate, s is the size of the drum, M is the maximum drum size, and m is the minimum drum size. This has the effect of making bigger drums have a lower playback rate (and thus a lower pitch), and smaller drums have a higher playback rate (and thus a higher pitch). We then play this new stream's sound. This entire process is essentially instantaneous after the user hits the drum.

One main limitation of the program's sound system is that it can't play multiple sounds at a time; each time a drum is played, it interrupts all other drum sounds that are still playing. Thus, fast or complex drum loops are essentially impossible to create with this interface. This is because the program uses Microsoft's native SoundPlayer interface, which doesn't support playing multiple sounds at once; some possible solutions would be to use DirectSound instead, or an open-source external library such as NAudio.

4 Evaluation

We wanted to see whether it was possible to interact with virtual drums via physical space in a meaningful way. More specifically, we wanted to see whether it was possible to create an experience where the user could create a drum, resize it according to what he/she wanted, and seamlessly play it. We expected it to be possible but not perfect, especially regarding the controls. The reason was because, after initial research, we realized that the Kinect doesn't readily support fine-grained tracking of hand gestures. We would have needed to apply some sort of machine learning algorithm to classify the different gestures. However, due to time constraints we knew that this wasn't feasible. In fact, having a working 3D version was our stretch goal.

After some playtesting, we found our hypothesis to be correct. Due to the absence of hand gesture detection, moving a drum often interfered with playing and resizing it. Another thing we found out was that our mechanism for releasing a drum wasn't intuitive. Many users experienced difficulty with this. Another, more subtle flaw we found was that not being able to output multiple sounds simultaneously significantly decreased the playing experience. We concluded that hand gesture detection and multiple sound output are a must, and these should be two essential features of our application.

5 Conclusion/Future Study

We created a viable solution that allows the user to create, modify and play drums. This is a good first step towards a virtual user-generated drumset. We believe that our model can also be applied to other types of drums and instruments; this could potentially allow other users to join together and create a band of interactive instruments.

The ease of use of the UI is something that we plan to improve and test in the future. Right now, we only change the color of the drums based on the action performed. Including icons similar to cursor icons for each action will make it easier to tell which action the user if trying to perform. We can gather a random sampling of people and ask them to play our instruments along to the beat of a song. From there, we can measure what the difficulties of playing our drums are, and record the users accuracy and monitor frustration while using our platform. We will be able to tell what is intuitive and what is not and improve our software from there.

Our interface is an example of how we would like other interactive interfaces to work. We believe that by giving the user control of the UI, you can build a better user-friendly interface. We can see this functionality being implemented in not only Kinect games, but also all types of interactive software. These interfaces should give the user control over what they are doing in order for them to be able to work and play in a customizable environment.