# Using Movement to Navigate Through a File System Contained Within a 3D Environment

**Natalie Burke**

CIS4914, Senior Project

Adviser: Douglas D. Dankel II, *email*: ddd@cise.ufl.edu

University of Florida

Department of CISE

Gainesville, FL 32611

**Date of Final Presentation: April 22, 2011**

## 1. Abstract:

This project focuses on the use of motion control to navigate a common everyday computer application. More specifically the creation of a new graphical user interface design for accessing file systems that can be successfully navigated using only the Microsoft Kinect. The goal here is to create a design that utilizes the Kinect's ability to understand depth and space and be able to perform common operations using only skeletal tracking. Through careful planning and several iterations the best design appears to be that of a ring. A ring can be quickly traversed and can show a direct relationship between files and directories. The best means of utilizing skeletal tracking is by comparing joint locations at certain times. For example the left hand being higher than the right will select the file the user is currently closest to. For the most part the movement choices made for different operations are easy to learn and intuitive to the design decided upon. However, issues still exist with using motion capture. The depth maps created are noisy and precise movements are nearly impossible. Even so the increased accessibility of motion capture devices is elevating their importance in the role of future computer applications.

**2. Introduction:**

Two major problems exist:

1. How does the ability to perceive depth change current computer navigation structures?

2. How does one use skeletal tracking to perform common actions?

The increasing application of motion control for common computer usage creates a whole new set of problems that must be addressed; a new way to organize computer fundamentals in three dimensional space must surface. Two dimensional programs ignore the major advantage motion control has over the current mouse and trackpad: an ability to understand depth and movement in space. With motion control the desktop environment is truly an environment existing in three dimensions. On a highly abstract level the desktop environment can be thought of as changing from a rectangle to a cube. This creates 5 new surfaces as well as the space between these surfaces. Common everyday computer applications need to be able to exist in this environment and fully utilize this new dimension of space.

My project specifically focuses on just one of these common computer application: the file management application. This is the graphical user interface (gui) used everyday for accessing the file systems. Beyond its organization and change from two dimensions to three there is one more major issue that must be addressed in designing a new gui. How to navigate through this three dimensional space only using skeletal tracking. Also due to the limitation of my specific hardware choices there exists the issue of how to use full body skeletal tracking without the ability to know the location of individual fingers. Furthermore, strong gestures should be avoided due to the range in the sizes of people that would need to be accounted for and inconsistencies of what different people consider a certain motion to be. All control must be connected to specific skeletal joints and actions should be interpreted from the relationships that occur between them

**Index:**

**3. Problem Domain:**

My specific project goal is to create a three dimensional gui that can be used to access a specific file system on my computer using only skeletal tracking. The design should be easy to understand, intuitive, and quick to navigate through. It should also take advantage of all possible dimensions and in some way incorporate the use of depth. The navigation through it should rely entirely on skeletal tracking. In addition these movements should make sense for the data structure being traversed and should feel natural.

Constraints exist due to my software and hardware limitations. For hardware this project uses the Microsoft Kinect as a basic motion capture device. The gui was created in OpenGL and the Kinect was integrated using PrimeSense middleware and the OpenNI APIs.

**4. Literature Search:**

4.1 Motion Capture, specifically the workings of the Microsoft Kinect.

Motion capture is the process of recording and digitally translating movement. The techniques used by the Kinect classifies it as a markerless motion capture system. An infrared projector projects a known pattern and creates a depth array of pixels by comparing the seen pattern and the known pattern. The depth map format is 320 by 240 resolution. Each pixel is described by 16 bits, the upper 13 containing the depth in millimeters while the lower 3 gives the segmentation mask (0-6 based on the current skeleton being tracked, 0-no player or unknown. Unknown results from areas in shadows, or of super low or super high reflectivity). The Kinect has three cameras, two used to create the depth map. One projects the infrared pattern and another reads the depth array to create the visual depth map seen in figure one. The third camera on the Kinect is a standard RGB camera, that when combined with the depth map renders a three dimensional image of the user from a single angle. Three Kinects can actually render a full 3D object.

The specific middleware component used in this project to communicate with the Kinect is PrimeSense's full body analysis software. This component processes sensory data and generates a data structure that describes the locations and orientations of each specific joint in space. Returning the data of a specific joint gives it's relative x,y, and z location, which in turn can be used to control the x,y, and

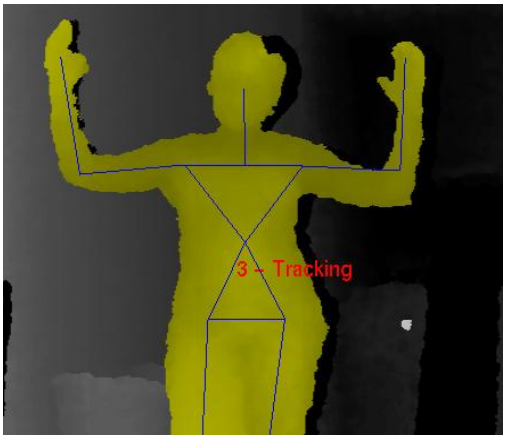


*Figure 2: Resulting Depth Map*

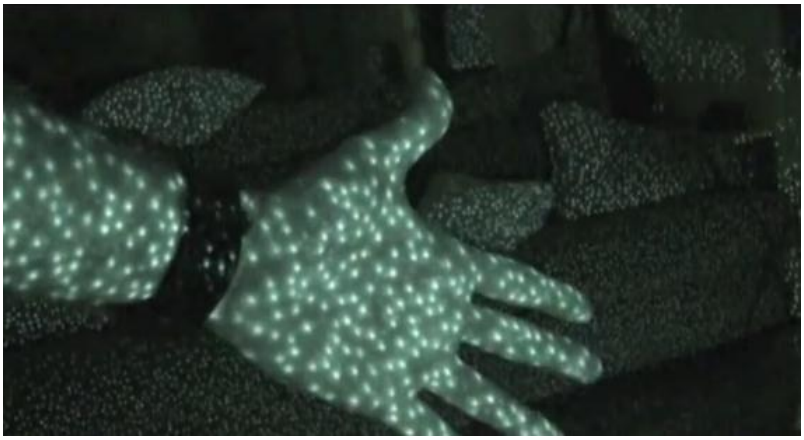

*Figure 1: Infrared projection*

z location of any created object. This data is what controls object's motions in Kinect projects.

4.2 Software Architecture used by the Kinect

At the heart of the Kinect is machine learning. Using a data base of known skeleton coordinates the Kinect is able to quickly calibrate a close to accurate 20 joint skeleton and remember it for future use. The OpenNI APIs are not as quick at performing this calibration as the Microsoft APIs since it does not relay on machine learning to remember the skeleton's of various past users. The pipeline architecture for Kinect projects all follow the same basic outline. Using the depth stream the player is found, the background is removed and the skeleton tracking starts. Only after those three requirements have been matched are any graphics updated.

Another important issue that must be addressed when using motion control is Skeletal Filtering. Many factors can cause interference and depth maps are extremely noisy. Small frequency jitters and temporary spikes must be filtered. These jitters are most apparent in the hands and feet. If left unfiltered these jitters can cause the extremity joints to unnaturally flip in orientation, which creates a big issue when using the Kinect to animate characters. The three main filtering algorithms used when writing applications that use motion capture devices, are Bayes Filtering, Gaussia Filtering, and Particle Filtering. Also important to remember when using motion control is to give the user feedback to their gestures and movements. The user will quickly learn and adjust to the sensitivity of the specific application.

**5. Solution:**

5.1 A review of the problem:

1. How does the ability to perceive depth change current computer navigation structures?

2. How does one use skeletal tracking to perform common actions?

Described here is the original solution created to solve the problems above. Upon further iterations some of these plans did change based on the surfacing of new problems. These changes are discussed in the results section.

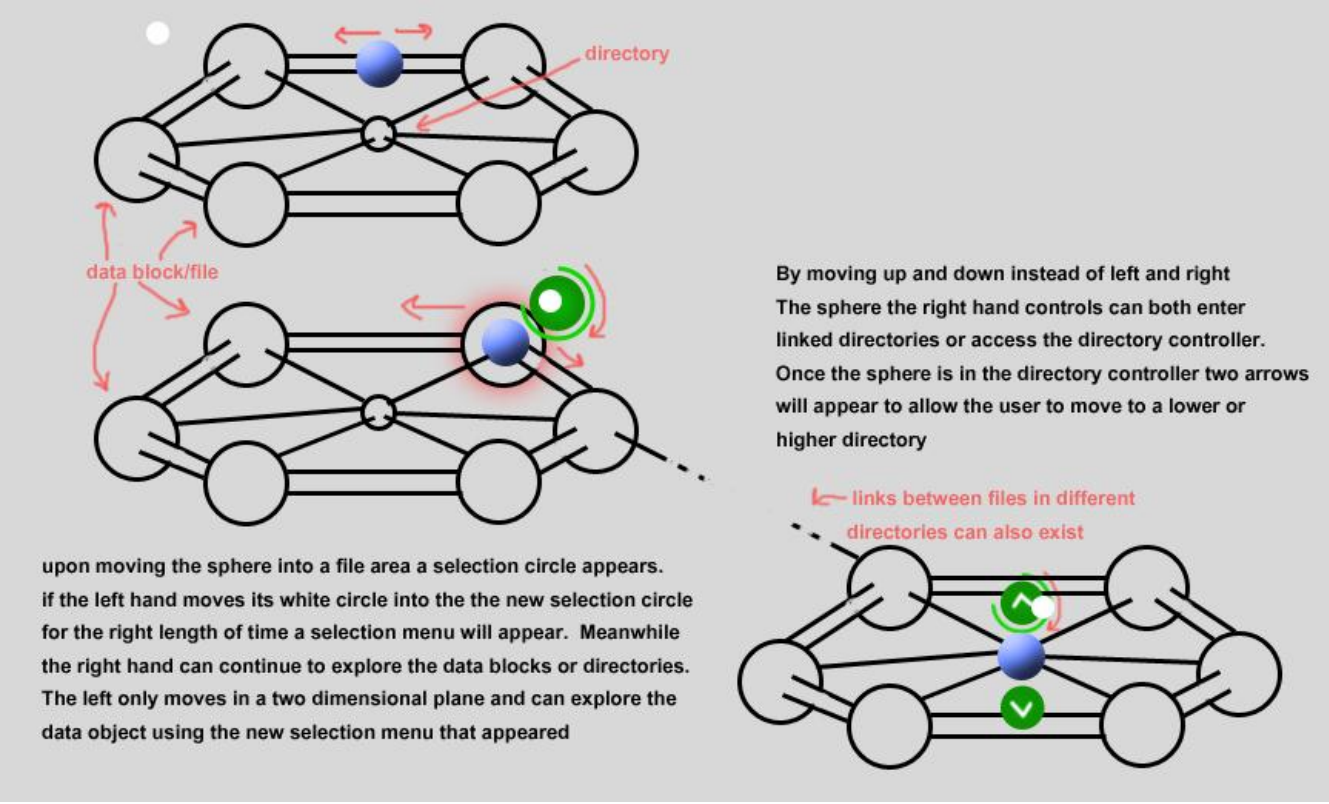


*Figure 3: Initial Design Solution*

5.2 The Design

Due to the complexity of controlling where an item is in three dimensions it was best if all files lie along a type of rail system. Files from one directory needed to have some relationship to each other and directories should be easily distinguishable from files. The initial design solution is shown in Figure 3. The files are organized in a ring around a central directory block, and other connected

directories are accessible by accessing the central directory block. Opening a file is done by locking onto a file block and hovering over some selection object for a specific amount of time. The left hand controls a cursor that is used to make the selections. The selection options appear around the file blocks they are associated with.

2. Navigation

Navigation through the gui needs to feel intuitive. Directory motions and file navigation need to make logical sense with the design created. Initially gestures were to be completely avoided. The right hand controls a sphere that moves between the files when moving right and left and moves between different directories when moving up and down. The left hand controls a cursor used to choose the file operations.

**6. Results:**

During the iterations of this project many of the initial solutions changed. The importance of gestures were realized and some difficulties of using the initial design of the file manager application were discovered. In the final iteration the files are organized in alphabetical order around their central directory and the options for each file are no longer situated around the file block itself but rather in a shared menu system that lies in the foreground. This menu system allows for opening and deleting files as well as rotating the gui. The biggest reason for this change was using a cursor with the left hand while navigating with the right was a grueling task. It became very difficult to keep the right hand still while the left hand was moving to various options. Also noise in the Kinect's depth map cause locking onto specific file blocks quite difficult. It was also very easy to accidentally change directories and open the wrong files without a successful locking mechanic.

The only solution for many of these problems was to figure out a way to use gestures effectively and consistently. The first method is to use the relationship between joints to signal a motion. This is a simple solution to program and quick for most users to understand and effectively take advantage of. The other option was to use include invisible solids in specific locations of the gui environment and use collision detection to realize when a movement cause a collision with these solids. Although this solution is harder to program there aren't any limits of what is considered a motion. Due to the time constraints this project uses joint relationships.

6.1 The final navigation plan can be described as follows:

The file system gui keeps track of the location of five joints: the right hand, the left hand, the left elbow, the head, and the waist. The right hand controls the movement in between files. To lock onto a specific file the left hand must be raised higher then then right. Upon locking onto a file, the file can then be opened or deleted, or the entire gui can be rotated. When hovering over an item in the menu a timer starts and after three seconds holding the left hand in a relatively constant location that action will take place. To move up a directory the right hand must move above the user's head. If more than one directory can be traveled to their names will appear in a menu format similar to that given when locked onto a file. To move down a directory the right hand joint must be positioned to the left of the left elbow joint and to leave the directory space the right hand must be below the waist. The final

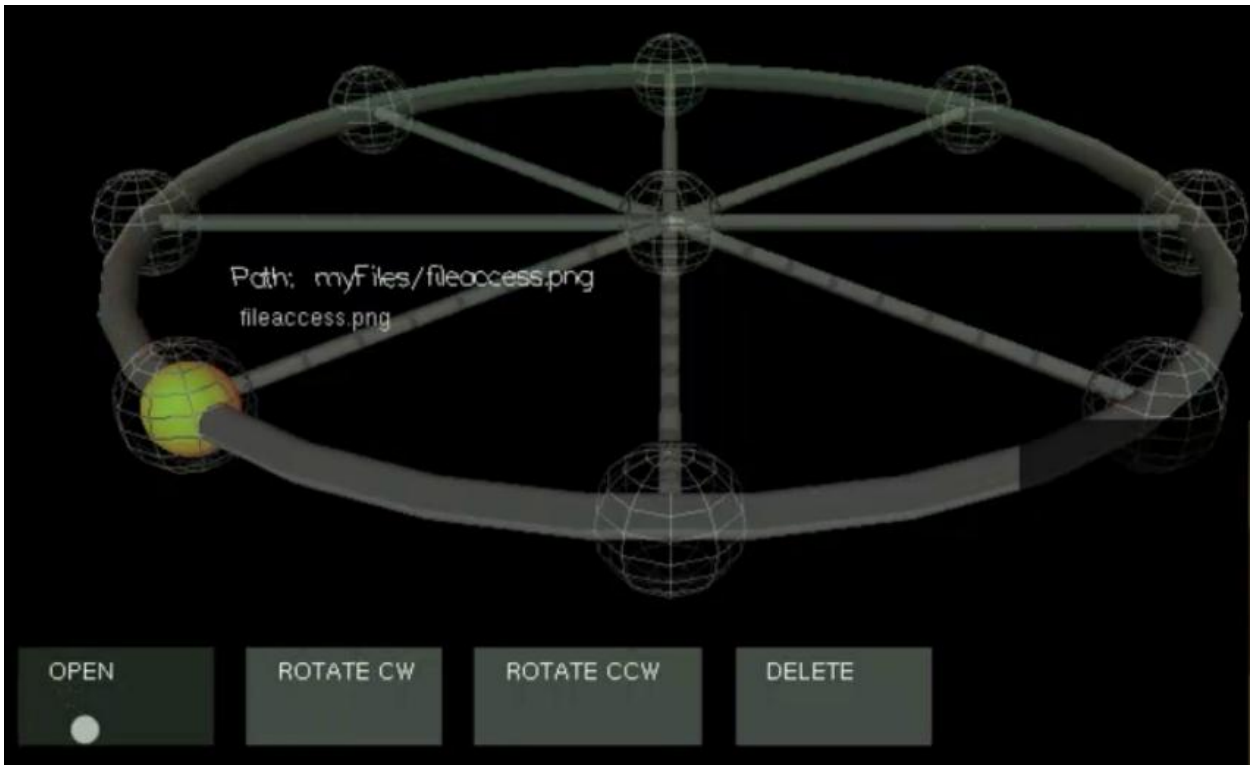


*Figure 4: Design after final iteration*

design is shown in figure 4.

**7. Conclusion:**

Motion capture is an extremely interesting field that is quickly increasing in it's capabilities The

Kinect is an amazing piece of hardware. It is a low priced, full body, markerless motion capture device. With it's release a myriad of communities have appeared full of people working on projects that utilize the technologies of this hardware. Applications that would have seemed unrealistic some years ago are now relatively easy and inexpensive to produce. Later this spring Microsoft will release it's developer tools for the Kinect to the public and there have been announcements of the use of motion control in the next Windows operating system. Motion capture is a big part of the future of computing and I plan to research different designs for other operating system fundamentals. Further projects will be creating a three dimensional mouse driver and expanding this three dimensional network concept to encompass other computer applications.

**8. Acknowledgments:**

I would like to thank my adviser Douglas Dankel for brainstorming design ideas with me. I would like to acknowledge the OpenNI and PrimeSense forums as great places to learn more about writing applications that use the Kinect. Lastly I would like to thank Zolt Mathe for speaking at GDC about the Microsoft Kinect's use of skeletal tracking.

**9. References:**

Davis, T., Neider, J., Shreiner, D., Woo, M. (6th edition). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1*. Addison-Wesley Publishing

Kesson, Malcolm, (2002). *CG References & Tutorials.* Retrieved from http://www.fundza.com/index.html

Mathe, Zsolt. "Kinect-Skeletal Tracking" Reading. Moscone Center. San Francisco, California. 3 March. 2011.

OpenNI Organization. (November 2010). *OpenNI User Guide.*

PrimeSense NITE. (2010). *The PrimeSensor NITE 1.1.*

Swiftless Tutorials, Game Programming and Computer Graphics Tutorials. (2010). *OpenGL.* Retrieved from http://www.swiftless.com/opengltuts.html

Kinect Hacks, Hacking Guides, News and Downloads. (2011). Retrieved from http://www.kinect-hacks.com/

## 10. Appendix A:

Basics of setting up a Kinect project:

```
...
#define SAMPLE_XML_PATH "config.xml"
#define POSE_TO_USE "Psi"
…
xn::UserGenerator g_UserGenerator;
...
void XN_CALLBACK_TYPE User_NewUser(xn::UserGenerator& generator, XnUserID nId, void* pCookie) {
	printf("New User: %d\n", nId);
	g_UserGenerator.GetPoseDetectionCap().StartPoseDetection(POSE_TO_USE, nId);
}

void XN_CALLBACK_TYPE User_LostUser(xn::UserGenerator& generator, XnUserID nId, void* pCookie) {}

void XN_CALLBACK_TYPE Pose_Detected(xn::PoseDetectionCapability& pose, const XnChar* strPose, XnUserID nId,
void* pCookie) {
	printf("Pose %s for user %d\n", strPose, nId);
	g_UserGenerator.GetPoseDetectionCap().StopPoseDetection(nId);
	g_UserGenerator.GetSkeletonCap().RequestCalibration(nId, TRUE);
}

void XN_CALLBACK_TYPE Calibration_Start(xn::SkeletonCapability& capability, XnUserID nId, void* pCookie) {
	printf("Starting calibration for user %d\n", nId);
}


void XN_CALLBACK_TYPE Calibration_End(xn::SkeletonCapability& capability, XnUserID nId, XnBool bSuccess,
void* pCookie) {
	if (bSuccess) {
		printf("User calibrated\n");
		g_UserGenerator.GetSkeletonCap().StartTracking(nId);
	}
	else {
		printf("Failed to calibrate user %d\n", nId);
		g_UserGenerator.GetPoseDetectionCap().StartPoseDetection( POSE_TO_USE, nId);
	}
}

…
void display(void)
{

/**KINECT CONTROLLING CODE**/
	XnStatus nRetVal = XN_STATUS_OK;
	xn::Context context;

	xn::EnumerationErrors errors;
	nRetVal = context.InitFromXmlFile(SAMPLE_XML_PATH, &errors);
	nRetVal = g_UserGenerator.Create(context);
```

```
	XnCallbackHandle h1, h2, h3;
	g_UserGenerator.RegisterUserCallbacks(User_NewUser, User_LostUser, NULL, h1);
	//DETECTS THE USER
	g_UserGenerator.GetPoseDetectionCap().RegisterToPoseCallbacks( Pose_Detected, NULL, NULL, h2);
	//SEARCHED FOR THE CORRECT POSE
	g_UserGenerator.GetSkeletonCap().RegisterCalibrationCallbacks( Calibration_Start, Calibration_End, NULL, h3);
	// RETURNS USER AS READY FOR CALIBRATION
	g_UserGenerator.GetSkeletonCap().SetSkeletonProfile( XN_SKEL_PROFILE_ALL);
	//BUILDS THE SKELETON
	nRetVal = context.StartGeneratingAll();
	//STARTS TRACKING THE USER

	while(TRUE){
		//IF THE USER IS BEING TRACKED UDATE ENVIRONMENT
		nRetVal = context.WaitAndUpdateAll();
		XnUserID aUsers[15];
		XnUInt16 nUsers = 15;
		g_UserGenerator.GetUsers(aUsers, nUsers);
		for (int i = 0; i < nUsers; ++i) {
			if (g_UserGenerator.GetSkeletonCap().IsTracking(aUsers[i])) {
				XnSkeletonJointPosition Right_Hand;
				XnSkeletonJointPosition Head;
				XnSkeletonJointPosition Waist;
				XnSkeletonJointPosition Left_Elbow;
				XnSkeletonJointPosition Left_Hand;

g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition( aUsers[i], XN_SKEL_RIGHT_HAND, Right_Hand);
g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition( aUsers[i], XN_SKEL_HEAD, Head);
g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition( aUsers[i], XN_SKEL_WAIST, Waist);
g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition( aUsers[i], XN_SKEL_LEFT_ELBOW, Left_Elbow);
g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition( aUsers[i], XN_SKEL_LEFT_HAND, Left_Hand);

/**EXAMPLE OF LOCKING ON TO A FILE BLOCK**/
	if(lock==0){
		currentY=(int)abs(Right_Hand.position.X)%360;
	}
	for(int i=0; i<FILES; i++){
		if(currentY>=(i*(360/FILES))-2&&currentY<(i*(360/FILES))+2){
			lock=1;
		}
	}
	if(lock==1&&Right_Hand.position.Y>Left_Hand.position.Y){
		lock=0;
	}
}
context.Shutdown();
}
```

**11. Biography:**

Natalie Burke was born March 19, 1989 in Tampa, Florida. She attended high school at H.B. Plant and is currently completing her bachelor's degree in Digital Arts and Sciences at the University of Florida. She plans to graduate May 2011 and pursue a career as a technical artist. She has a great interest in new technologies and their integration into the fine arts. Outside of school she enjoys drawing. playing saxophone in a local band and long strolls on moonlit beaches.