COSC 470
Final Report

Kevin Shen
Supervisor: Geoff Wyvill

October 2012

Contents

1. Abstract

The objective of this project was to create a virtual sculpting tool that uses 3D input. The application uses the Playstation Move controller as a physical input device and the Move.Me software to obtain 3D coordinates of the controller. The Move.Me software is run on the Playstation and supplies 3D coordinates via wireless connection to the client, which renders the sculpture as a trail of spheres. A 3D input system eliminates problems inherent in traditional computer sculpting tools, such as converting 2D input to 3D coordinates. The application uses several forms of visual cues to enforce the illusion of three dimensional sculpture and aid in the user's hand-screen coordination. I demonstrate that 3D sculpting applications may become more intuitive and have fewer limitations with true 3D input.

2. Introduction to 3D Sculpture

This will discuss other research projects involving 3D sculpting, including Ross's project and its limitations (helix problem). Also discuss the previously high cost of motion tracking systems versus the relative low cost of consumer products like the PS3.

3D sculpting is a form of 3D modelling. The intent of sculpting is to provide a more natural, intuitive way to model. A common schema for 3D sculpting applications is to allow the user to manipulate a virtual material such as clay by creating, removing, squashing, or stretching. Parent [2] was the first to create a sculpture application. It allowed the user to cut and join geometric primitives to form sculptures. Galyean and Hughes [1] created a sculpting application of finer resolution in 1991. It consisted of four tools: one for removing material, one for adding material, one for "sanding" down material, and one for "melting" material. A relatively fine level of control in adding, deleting, and rendering material was possible due to their use of a voxmap, the 3D analogy to a 2D bitmap. An explanation of how a voxmap works can be found in [1].

Many 3D sculpting applications also incorporate some form of 3D input instead of traditional 2D mouse input. Despite extensive study on input devices, no single ideal device has been found [3]. Galyean and Hughes [1] used the Polhemus Isotrak, a controller that reports the so-called six degrees of freedom: X, Y, and Z position as well as three angles representing the orientation of the controller. Parviainen et al. [3] used the Magellan Space Mouse, a device that sits on the tabletop and also gives six degrees of freedom. More recently, motion tracking systems associated with video game consoles have become easily available. Vinayak et al. [5] used the Xbox Kinect, which tracks motion of the whole body, in their pottery application. In this case, sculpting was purely additive.

In this project I attempted to create a sculpture application that uses a controller (the Playstation Move) that is more analogous to a real-life sculpting tool. The application is based on Ross Phillips' [4] Octagon sculpting program in that its main purpose is creation of geometric primitives on a rotating turntable. I attempt to create a sculpture program that is intuitive and natural to use by combining the simple creation action with a true 3D input controller.

3. Playstation Move controller and Move.Me software

Sony markets a relatively cheap motion-tracking system for use with their Playstation3 video game console. The system consists of the Move controller, the Eye camera, and Move.Me software that creates a server on the Playstation. The controller features a coloured ball at one end, which is used by the Eye camera to track the position of the controller. The controller also has a trigger and six buttons. See Figure 3.1 for the structure and button layout of the Move controller.

Figure 3.1: Playstation Move controller

The software is capable of reporting six degrees of freedom. The software communicates with a client via a local wireless network. Data is represented as multiple states. This includes, among others, state of the controller, camera, and current camera frame. Information about button presses and controller position are included in the controller state.

4. Work done

4.1 Selection and setup of tools

The current market offers many consumer motion-tracking systems. Most are associated with current-generation game consoles; each of the Wii, Xbox 360, and Playstation3 has motion-tracking peripherals. I chose to use the Playstation's Move controller because of its availability and precision. The Otago Graphics Research Group was already in possession of a Playstation3, Move controller, and the Move.Me software. The motion data reported by the Move.Me is specific to the controller, whereas the Xbox's Kinect motion tracking system tracks motion of the entire body, resulting in less precise reports with unnecessary data. The Wii has minimal published support for developers wishing to track motion of the controller. Thus, the Playstation Move was the best choice for this application.

The Move.Me software is supported for Linux and Windows in C and C#. I intended to use the C libraries in a C++ project on Linux, but encountered difficulties with interleaving C and

C++ code. The Linux workstation also lacked appropriate wireless drivers. Finally, the C# Move library is able to easily connect to the Playstation server whereas the C library is not. Given the project's time constraint of one semester, I switched to a C# environment on Windows to gain access to the connection functions in the C# library. Once the project platform and language were chosen, both the Playstation3 and workstation were connected to a wireless router to allow data transfer between the Move.Me software and client.

4.2 Sculpting with the Move controller

The application has a museum scene in which the user can create sculpture (see Figure 4.1). Sculpting can be controlled using only the Move controller. The application is calibrated such that the user should stand approximately 1.5 metres from the Eye camera for a full range of motion. A coloured sphere indicates the position of the sculpting tool in virtual space.

To create or add to the sculpture, pull the trigger and move the controller.

To rotate the turntable and the sculpture clockwise, press the Cross button on the controller.

To rotate anticlockwise, press the Circle button.

To delete the current sculpture, press the Triangle button on the controller.

To switch between the full screen window and the GUI, press the Square button.

Figure 4.1: The empty sculpture scene

4.3 Rendering the sculpture

The sculpture is rendered as a series of spheres. The position of each sphere is stored in a list. Each time the state of the controller is updated, the application compares the current position of the sphere to the location of the last sphere that was added to the list. If the two spheres are greater than a certain distance apart and if the controller state indicates the trigger is being pressed, the sphere is added to the list of spheres to render.

Because the Move.Me software reports the 3D coordinates of the controller relative to the Eye camera, almost no calculation is needed to determine where to render the sphere on the screen. Given the Eye camera exists at position (0, 0, 0), the detectable range of motion for the controller is between -700 and 700 on the X axis, -450 and 450 on the Y axis, and 100 and 1800 on the Z axis. See Figure 4.2 for reference. Note these are the X and Y limitations when the user stands approximately 1.5 metres from the Eye camera; because the Eye camera's vision is limited to a frustrum, the X and Y coordinates that bound the area of detectable motion vary based on the Z coordinate.

The application simply creates an OpenGL scene with the camera at approximately the same location as the user's eyes. The centre of the workstation screen is considered (0, 0, 0) because the avatar of the sculpting tool cannot extend out of the screen (in the negative Z direction). The height of the user's eyes above the centre of the workstation screen depends on both the height of the workstation desk and the height of the user. Because this varies, the application is currently calibrated using a workstation screen height of 1m and user of 175 cm.

The arbitrary units reported by the Move.Me software are the units used by the OpenGL scene. By experimentation, one centimetre is equal to seven arbitrary units. Given the user stands 1.5 metres from the screen and his or her eyes are 30 centimetres above the screen midpoint, it is a simple conversion to place the OpenGL camera at (0, 210, -1050).

With the camera appropriately placed, it is a simple matter to render the spheres at the position reported by the Move.Me software.

Figure 4.2: Coordinate system of the scene

4.4 Rotating the sculpture

The turntable and sculpture can be rotated to view the sculpture from any angle. The position of the centre of the turntable is a known constant. The vector between the origin and the centre of the turntable is also known. Sphere rotation is implemented by translating the sphere by the negative of this vector. This translates the sphere toward the origin. The sphere is then rotated about the origin and translated by the positive of the position vector to return it to its correct position relative to the centre of the turntable.

4.5 Visual cues

A number of visual cues are used to provide reference for position and enforce the illusion of 3D.

The backdrop of the scene provides context for the 3D position of the sphere. Grid lines on the floor and turntable give the illusion of perspective and help the user track the X and Z position of the avatar. The backdrop was created by Ross Phillips [4] for his Octagon project.

The position of the drop shadow under the avatar's sphere helps the user understand where the avatar is in the X and Z directions. The size of the shadow, which grows as the avatar approaches the ground, provides context for the height of the avatar (see Figure 4.3).

Finally, the rotation function allows the user to examine the sculpture from any angle. This allows the user to better understand the position of each sphere relative to all the other spheres. Rotation is illustrated in Figure 4.4.

Figure 4.3: Difference in shadow size

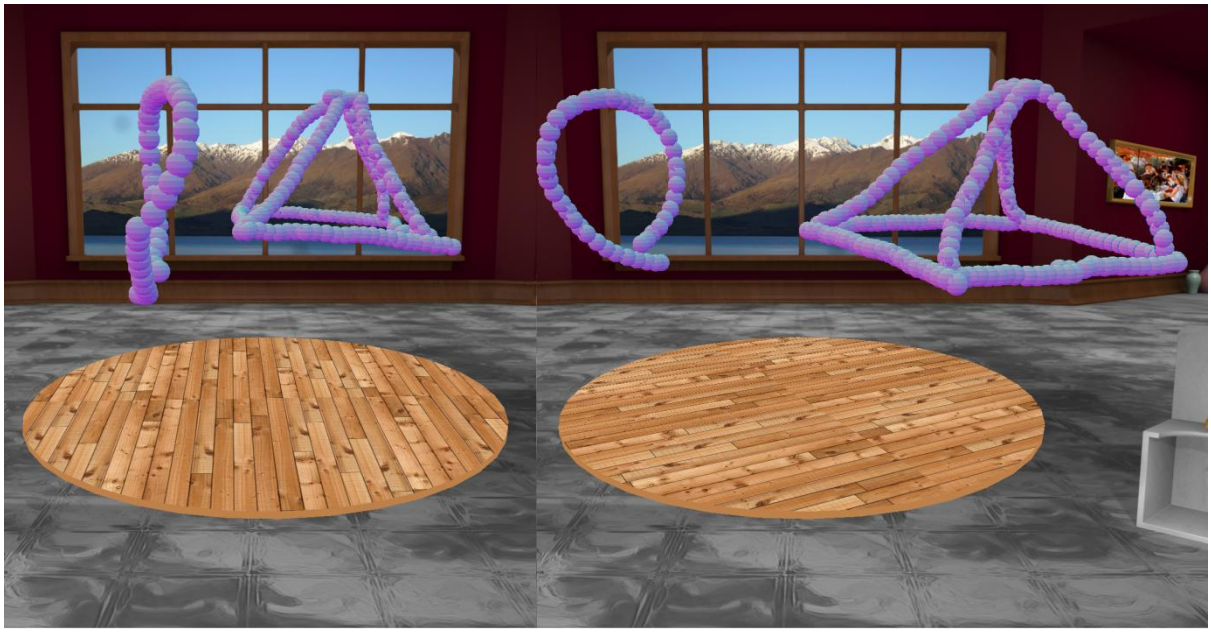

Figure 4.4: The same sculpture rotated clockwise

5. Discussion

5.1 2D to 3D ambiguity

The use of position tracking software in this project eliminated ambiguities associated with traditional 3D modelling and sculpting programs that use 2D input. In these applications, a mouse click with 2D screen coordinates in X and Y must be converted to a set of 3D world coordinates in X, Y, and Z. One way to do this is by treating the screen as a plane called the view plane, which exists in 3D space in front of the scene's camera. A ray is cast from the camera and travels through the coordinate of the clicked point on the view plane. The ray continues on through to the scene and may intersect existing 3D objects in the plane.

This method is suitable for tasks such as selection of visible objects. The first object that intersects the ray is the object that is closest to the camera and thus the closest visible object. However, ray casting is a poor method for object creation. In a blank scene, it is utterly ambiguous which 3D world coordinate the user intended to indicate. Any coordinate on the ray between the view plane and the end of the scene is valid. Thus, an object cannot be created in a blank scene with a single click, because 3D coordinates cannot be determined from a 2D mouse click coordinate without additional information.

Phillips [4] addressed this issue by requiring the user to create primitives at an intersection with an existing object in the scene. Valid intersection objects included user-created primitives or a turntable on the floor of the room in the scene. The tube, the primitive that most closely resembles the sphere trail, was created with a single mouse gesture. The end points of the gesture had to intersect with objects in the scene. From these endpoints, a vertical plane was constructed that contained both endpoints. All points of the tube were then created in this vertical plane. Thus, tubes in the Octagon project were always coplanar. Constructing a non-coplanar object like a helix from the tube primitive was impossible without additional objects to serve as anchor points. The 3D position reported by the Move.Me system allows for helix creation without these anchor points.

5.2 Intuitive movement

Without Phillips's [1] intersection method, current 2D-input 3D modelling applications like Maya default to object creation at the origin. The user can specify 3D coordinates at which to create basic 3D shapes (primitives) by manually entering world coordinates. This method of creation takes 8 mouse clicks and keystrokes in Maya. Because this method is unwieldy, users often choose to create primitives at the origin and then scale, rotate, and translate the primitive as necessary. Creating a unit cube and translating it along 3 axes takes 6 mouse clicks in Maya.

The Move system provides 3D coordinates of the controller to the client. With true 3D input, creation of primitives at a specific world location can be done with a single button press. However, users of 3D input must be able to coordinate their real position with screen position. This is done with the use of an avatar; the controller's position is represented on the screen as a differently-coloured sphere. The user should be able to move the physical controller and see the avatar move as they expect on the screen.

For movement to be intuitive, the illusion of 3D must be enforced. The primary way of doing this was with the background. I initially intended to mirror the real world in the virtual world by using video frames from the Eye camera as the background. This failed because the video frames provided by the Move.Me software were too low-resolution to give any meaningful visual cues and took too long to render in GL despite their resolution. Instead, I used the backdrop from Ross Phillips's Octagon project [4].

With a static background the avatar can either mirror or emulate the user's movements. In mirrored movement, the avatar grows as the controller gets closer to the screen, as it would in a real mirror. In emulated movement, the virtual space is a direct mapping of the real space. The avatar shrinks as the controller moves closer to the screen.

I chose to implement mirrored movement because of the placement of the Eye camera at the base of the workstation screen. In this setup, the space seen by the Eye camera is a mirror image of the virtual space on the screen. The virtual sculpting space can be thought of as a scaled version of the real space seen by the camera, reflected across the plane of the screen. Figure 5.1 illustrates this.

Additionally, implementing emulated movement with the camera facing away from the screen presents the problem of a clipped frustrum. The Eye camera's view is limited in the X and Y directions when objects are near the camera, where the frustrum has the smallest cross-section. Moving the controller closer to the camera moves the avatar farther back in emulated space and away from the view plane (screen). The result is restricted movement in the X and Y direction as the avatar moves farther back in the scene. See Figure 5.2 for an illustration of this.

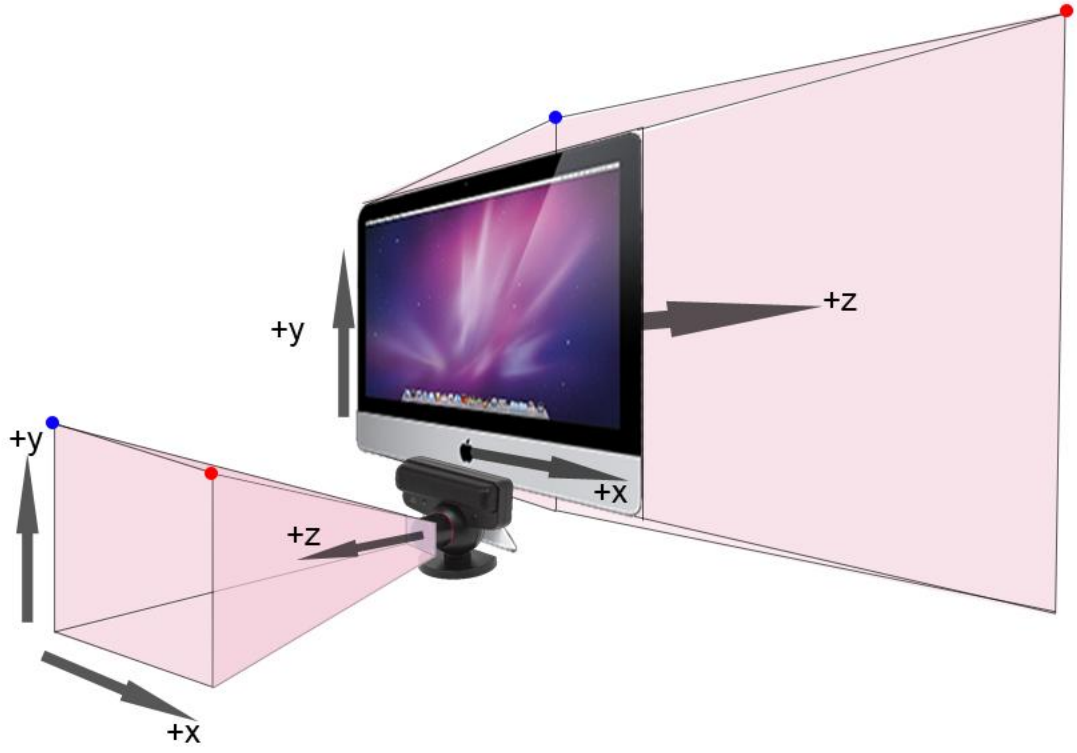

Figure 5.1: Mirrored frustrums. The red and blue dots show which corners map to each other.

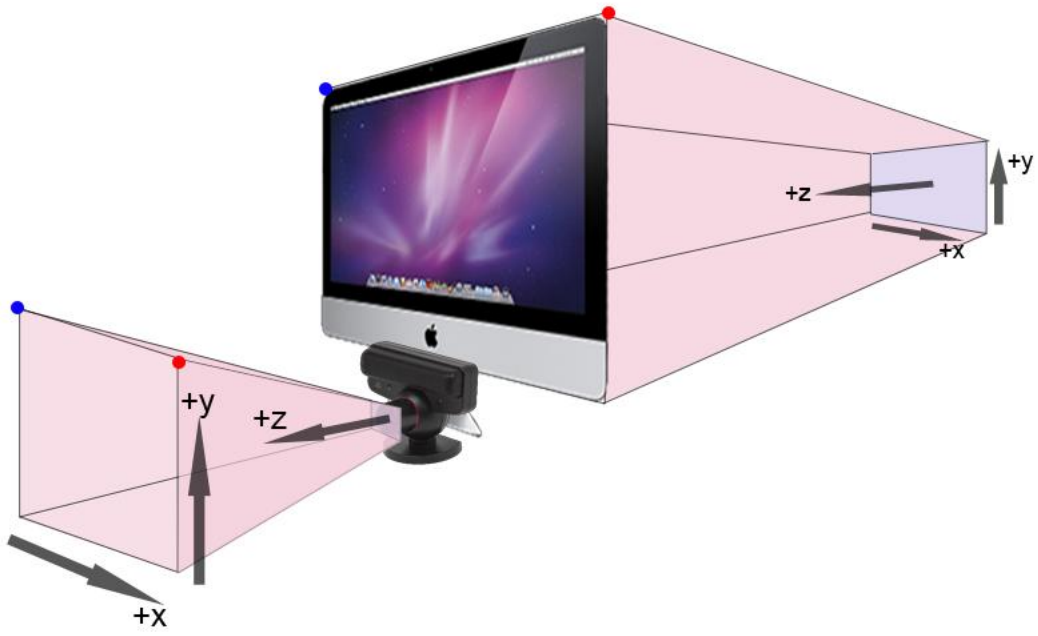

Figure 5.2: Clipped frustrum encountered in emulated movement

6. Future work

Though the application is relatively intuitive, new users still require instruction. Rotation could be made more intuitive by moving the avatar to hooks or latches on the edge of the turntable to toggle rotation and allowing rotation to be controlled by gestures of the controller.

The sculpting function itself is limited. Many of Phillips' [4] or Galyean's [1] features could be implemented to increase the range of tools available to the user. Sculpting is currently purely additive. Allowing for subtractive sculpting by deletion of primitives would improve the precision with which the user could create 3D forms. Greater flexibility with primitives could also be implemented; allowing creation of additional primitives such as cylinders and cubes, as well as allowing the user to adjust the size of the primitive being created, would increase precision. Allowing users to select colours by moving the avatar to paint buckets would be highly intuitive.

The current calibration is hard-coded for a certain user distance and height. Allowing the user to calibrate when the application is launched would make movement more accurate and intuitive. This could be accomplished by having the user register the controller position when it is in front of their eyes to determine the OpenGL camera's position. The sculpting area could also be defined by the user to calibrate sensitivity of movement.

Like Galyean and Hughes [1], I do not use the orientation of the controller. Using the orientation of the controller would allow for more refined gestures. For example, an upside-down controller may represent a chisel that removes material, while an upright controller may represent a putty knife that applies material to the sculpture by creating material.

7. Conclusion

The main focus of this project was to demonstrate that the Playstation3 Move system is a usable 3D input device. I have achieved that with a basic 3D sculpture application controlled entirely by the Move controller.

Informal tests with new users showed the movement of the avatar was not as easy as desired to coordinate with movement of the controller. This is likely because of imprecise calibration. One source of error is the height of the user; allowing the user to calibrate the application as discussed in "Future Work" would reduce this error. Another source of error is the unit conversion itself. Conversions were done by experimentation. If the actual conversion could be obtained from the Move.Me server source code, this application's avatar would correspond to the user's movements more closely.

Initial goals of creating a subtractive sculpture application were not met. This was due in part to changes in the language and operating system for the project and unexpected delays in getting the Move.Me software running and connected. It took additional time to familiarise myself with C#, OpenGL, and the undocumented Move library. Much time was spent learning graphics principles and how to use these tools. As a result of these delays, I did not have time to complete the subtractive sculpture feature.

8. References

[1] Galyean, T. and Hughes, J. 1991. Sculpting: An Interactive Volumetric Modeling Technique. *Computer Graphics*, 25(4):267-274, July 1991.

[2] Parent, R. 1977. A System for Sculpting 3-D Data. *Computer Graphics*, 1(2):138-147, July 1977.

[3] Parviainen, J., Sainio, N., and Raisamo, R. 2004. Perceiving Tools in 3D Sculpting. In *Computer Human Interaction: 6th Asia Pacific Conference, APCHI*, Rotorua, New Zealand, June 29-July 2, 2004, Masood, M., Jones, S., and Rogers, B., Eds.

[4] Phillips, R. 2011. Cooperative Sculpture. Dissertation. University of Otago.

[5] Vinayak, Murugappan, S., Piya, C., and Ramani, K. 2012. Handy-Potter: Rapid 3-D Shape Exploration Through Natural Hand Motions.

9. Appendix

9.1 Launching the application

The following conditions must be met to successfully launch the application:

- Both the Playstation and workstation are connected to the same wireless network
- The workstation is running Windows 7 and has Visual Studio 2010 installed
- The Playstation has the Move.Me software installed and is currently running it
- The Move controller is calibrated. Instructions on this can be found at http://code.google.com/p/moveme/downloads/list

The sculpture application is built on top of the GUI provided by the Move.Me library. To run the application, open the project in Visual Studio 2010 for C# and run the PSMoveSharpTest project.

At the launch screen, the user must first connect to the Playstation server. Enter the IP address and port number of the Playstation server into the fields in the upper left corner of the GUI (see Figure 3). The IP address and port can be found in the upper left corner of the Playstation display (see Figure 4). Establishing the connection will result in immediate continuous transfer of controller data from the Playstation server to the client.

To begin sculpting, click on the “Sculpture” tab. The application will launch a fullscreen window depicting a museum scene with a turntable, as seen in Figure 2. Once in full screen, the application is ready for sculpting. See 4.2 for sculpting instructions.

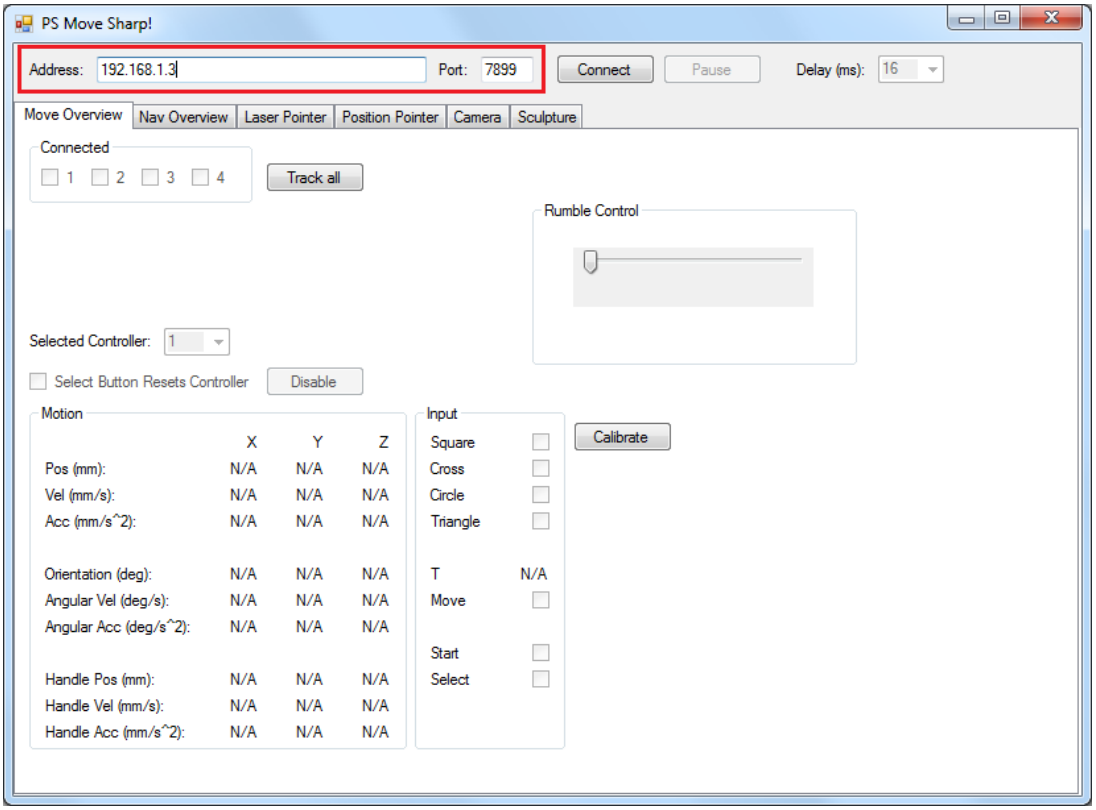

Figure 3: The application launch screen with server information fields highlighted

Figure 4: Playstation Move.Me display