SCKinect

Evan Murray
Aura Audio LLC
evan@auraaudio.io

ABSTRACT

SCKinect is a SuperCollider plugin that allows users to interact with a Kinect sensor. Its core implementation contains a unit generator called 'Kinect', designed to output motion-tracking data to control buses. The plugin also includes commands, facilitating interaction with Kinect devices through the interpreter. The interpreted nature of SuperCollider and the server-language duality allow multimedia enthusiasts to efficiently communicate with technical rendering systems. This is perfect for live performances and interactive installations. With the addition of this plugin, performers can interact with the Kinect directly in SuperCollider with low latency. This paper will cover the implementation of the plugin and its potential applications.

1. BACKGROUND

According to [1], the record for the fastest-selling gaming peripheral was set by the Kinect on January 3rd, 2011. The device sold on average 133,333 units per day in the first 60 days of its launch. Featuring a custom System on Chip, originally developed by PrimeSense, this device was designed to be good at detecting people and their surrounding environment [2]. In fact, it almost did this too well. Naturally, users became skeptical of the sensors built into the device and their privacy implications, as described in [3]. This, in combination with numerous other factors, led to the eventual decline of the Kinect as a consumer product over the next decade. According to [4], the last Kinect product (Kinect Azure) was discontinued in October 2023, marking over a decade-long life-cycle for this product.

2. MOTIVATION

If the Kinect was already discontinued and no longer being supported by its official creator, a valid question to ask is: why should one continue using it? Even better: how might one use it with SuperCollider? Musicians, whether they realize it or not, inherently control numerous parameters on their instrument–leading to a unique musical expression. To match the sound of a babbling crowd at a restaurant, a trombone player might mask their melody with a plunger mute to get that classic "wah wah" sound. In electronic music, a synthesizer performer or DJ might turn the knobs on their device which control the cutoff frequency of a highpass filter. This is typically done to prepare the crowd for a major transition in the song. In interactive programming languages, such as SuperCollider, performers don't have these knobs or tactile interfaces which physical instruments might offer. However, they can be implemented by connecting a joy stick or mouse as a Human Interaction Device. This will allow the performer to have some tactile input which they can use to control a particular parameter in their program, influencing the sound or visual output which is created.

3. NATURAL USER INTERFACES

There are a class of user interfaces which are encompassed under the term, Natural User Interfaces. These types of interfaces, according to [5], bypass conventional ones (such as the mechanical mouse and keyboard). [6] further classifies these as Reality-Based Interfaces (RBI). Something compelling about these is their aim to intuit real-world interactions. One of the themes mentioned in [6] related to RBI is body awareness and skills, which describes how "people have an awareness of their own physical bodies and posses skills for controlling and coordinating their bodies." This is what neuroscientists would call "proprioception." Virtual Reality devices take this into account with their head-mounted displays. Instead of interacting by keyboard or mouse, the user can tilt their head in different directions as they would in the real world.

4. KINECT INTERFACE

Using the Kinect and a couple of open-source helper tools, tracking a user's full body motion with high precision is possible. This is an alternative to using a VR headset with a tracker (required for full-body tracking in VR). Furthermore, the Kinect is much cheaper than the combined price of a VR headset and a tracker. Based on what was discussed previously in 2, this motion tracking tool can be used similar to a Human Interaction Device in SuperCollider, allowing the user to map body movements to control busses. A caveat to this, discussed in 5.2, is depending on the implementation–some sophisticated hardware may be required to perform with low latency. This typically requires a Graphical Processing Unit (GPU) to help the Central Processing Unit (CPU) with absurd amounts of data. The data comes from a pose estimation model which relies on machine learning algorithms. It's worth noting over the last decade, GPU's and CPU's have become smaller and more affordable–making their use a bit more practical to the average consumer. In addition, there are plenty of algorithms which don't rely on machine learning, such as the application of Dijkstra's algorithm demonstrated in [7]. These algorithms can perform low-latency human pose es-

timation on the CPU. A future goal of SCKinect is to incorporate algorithms similar to these for non-GPU users. However, for the sake of simplicity–the discussion below will focus on all the hardware and software components used in constructing the GPU-based machine learning approach.

5. HARDWARE COMPONENTS

The hardware components which made this project possible are described below. It's important to note the Kinect v2 needs a separate USB adapter for connecting to the computer, which costs around 20-40 US dollars. The Kinect v1 has an adapter which is sold for around half the price.

5.1 AMD Ryzen 5 3600 CPU

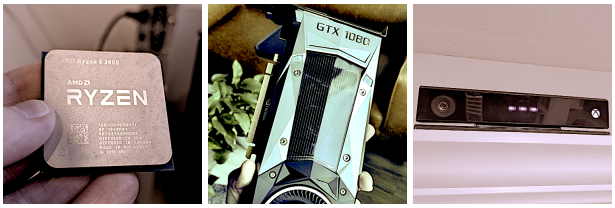
This CPU shown in Figure 1a was chosen for this project because it has a nice performance which doesn't cause the GPU to bottleneck. The Ryzen 5 3600 sells for about 40-80 US dollars with a base clock speed of 3.6 GHz and 6 cores with 12 threads.

5.2 NVIDIA GTX 1080 GPU

The GPU shown in Figure 1b was chosen for its compatibility with the pose estimation model. This is probably the most expensive hardware component of this project, selling for about 150-200 US dollars for a refurbished device. However, as noted previously–further options are being explored for CPU-only pose estimation. Additionally, NVIDIA's competitor–AMD–sells similar GPU's for about half of the price. Thus, it may be worth further investigation to explore adding support for other GPU brands as well.

5.3 Kinect

The Kinect v2, pictured in Figure 1c, is currently one of the cheapest Red Green Blue Depth (RGB-D) cameras on the market, selling pre-owned for about 50-100 US dollars. The Kinect v1 sells at about half the price with slightly lower specifications. At this time, SCKinect currently only supports 2D tracking. Thus, one technically doesn't need a Kinect and could use a camera instead. However, incorporating depth data into the existing pose estimation model is currently being explored. The Kinect v2 features a 1920x1080 resolution color camera, as well as a 512x424 resolution depth camera. The depth is based on the Time of Flight (ToF) sensors which put out infared light. It also has a 70x60 degree field of view and a 4-channel microphone array, which records audio at 48 kHz. There are also other methods of doing 3-dimensional (3D) pose tracking besides using an RGB-D camera (i.e. stereoscopic cameras). However, using an RGB-D camera as opposed to a stereoscopic setup may yield a higher resolution indoors, while stereoscopic cameras are better for use outdoors (since the sun's infared rays will overpower the Kinect's infared). One may also consider using multiple RGB-D and stereoscopic cameras for improved flexibility.



(a) Ryzen 5 3600 (b) GTX 1080 (c) Kinect v2

Figure 1: A picture of the three essential hardware components for the project.

6. SOFTWARE COMPONENTS

6.1 Libfreenect2

Libfreenect2, as shown in [8], is a library created by open-source developers for retrieving data from the Kinect. This data does not include pose estimations, but it includes all of the functions needed for interacting with the Kinect (i.e. opening, closing, and starting a stream of data).

6.2 OpenPose

OpenPose is the library responsible for returning the pose estimation data, pictured in Figure 2. It was developed by the Perceptual Computing Lab at Carnegie Mellon University and uses Convolutional Neural Networks to do markerless motion tracking. This tracking data includes 24 different joints on the human body. See [9, 10, 11, 12] for more information.

7. IMPLEMENTATION

SCKinect is implemented as a SuperCollider 3 plugin, featuring server commands for interacting with the components discussed in 6.1 and 6.2 respectively. The full source code is available on GitHub [1] . Currently, the plug-in works best if installed on Ubuntu 20.04. However, future releases will include support for both macOS and Windows–with or without a GPU.

8. COMMANDS

Commands allow SuperCollider to pass parameters from a SuperCollider class to the plugin, allowing one to configure the Kinect and OpenPose directly from SuperCollider. The currently implemented commands are explained below. They should also be run in the order they are shown when using SuperCollider.

8.1 `Kinect.findAvailable`

This is designed similar to the `HID.findAvailable` command in SuperCollider. It queries all of the Kinect devices connected to the computer and posts their serial numbers to the post window, as shown below:

```
-> Kinect
[Info] [Freenect2Impl] enumerating
devices...
[Info] [Freenect2Impl] 12 usb devices
```

[1] https://github.com/L42i/SCKinect

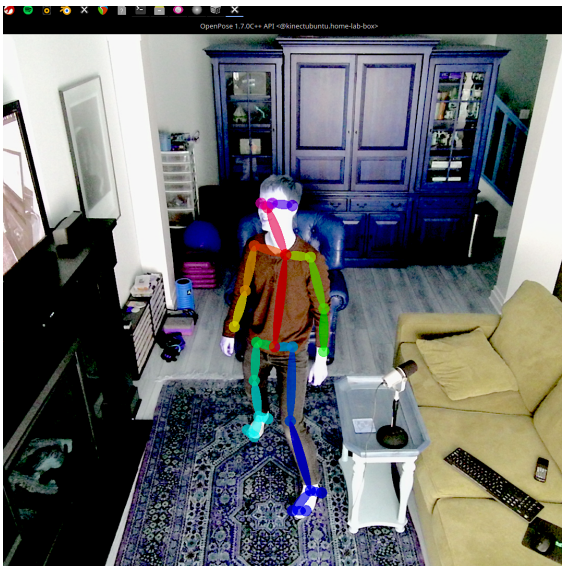


Figure 2: The window displayed in SCKinect which shows the 24 joints being tracked on the human body real-time.

```
connected
[Info] [Freenect2Impl] found valid
Kinect v2 @4:4 with serial
008953333347
[Info] [Freenect2Impl] found valid
Kinect v2 @4:2 with serial
065915234247
[Info] [Freenect2Impl] found 2
devices
```

8.2 `Kinect.setPipeline`

This command determines the device responsible for passing the data from the Kinect to OpenPose. The possible pipelines are Dump, CPU, OpenGL, CUDA, and CUDAKDE. For example, if one wanted to set the pipeline to use CUDA (assuming they have an NVIDIA GPU and the CUDA toolkit installed), they could run the command:

```
Kinect.setPipeline("CUDA")
```

8.3 `Kinect.openDevice`

`Kinect.openDevice` opens the device with the given serial number. For example, the second device shown in the output of running the command in 8.1 is opened by running:

```
Kinect.openDevice("065915234247")
```

8.4 `Kinect.start`

This command will tell `libfreenect2` to start preparing the Kinect for processing frames.

8.5 `Kinect.configureTracking`

This command configures the OpenPose model with specific settings. There are a lot of parameters which will be documented in footnote 1 Page 2. However, the most important one to understand is the path to the model folder, `"/home/emurray49/openpose/models"`. As opposed to the folder shown below, an existing path should be passed to the function instead. More instructions are available on GitHub Pages [2]. Here is what the full command might look like:

```
Kinect.configureTracking
(
3, 1,
"/home/emurray49/openpose/models",
1, 0, 1, 0.25,
0, "-1x-1", "-1x256",
1, "BODY_25", 0.5,
0.5, 0, 0.05, -1, 0.0
)
```

8.6 `Kinect.startTracking`

If the previous commands run successfully, one may start the motion tracking by running this command. Upon running this command, the pose tracking window will be displayed, as shown in Figure 2.

8.7 `Kinect.hideDisplay`

If one wishes to hide this display of the pose tracking, they may disable it by running the above command.

8.8 `Kinect.showDisplay`

To re-enable the pose tracking display, one may run this command after it has been disabled.

8.9 `Kinect.stopTracking`

This will stop the motion tracking and hide the display window if shown.

8.10 `Kinect.stop`

This command will stop the Kinect frames.

8.11 `Kinect.closeDevice`

`Kinect.closeDevice` closes the device with the given serial number. For example, the device shown in the output of running the command in 8.1 is closed by running:

```
Kinect.closeDevice("065915234247")
```

9. MAPPING

The `Kinect` UGen included in the SCKinect plugin includes a `.kr` method for mapping a specific joint to a control bus. Consider the following example of creating two control busses for a synthesizer:

```
~amplitudeBus = Bus.control(s, 1);
~frequencyBus = Bus.control(s, 1);
```

The first bus could be used to control the amplitude and the second bus could be used to control the frequency. The next goal is to map joints on the human body to these parameters. How about mapping the y position of the right

[2] https://emurray2.github.io/spatial-auditory-feedback

wrist to the amplitude and the x position of the left wrist to the frequency? Here is what that might look like:

```
var a = ~kinectAmplitudeBus.index;
var b = ~kinectFrequencyBus.index;
var e = "RWrist";
var f = "LWrist";
var c = Kinect.kr(0, 0.5, e, "Y"));
var d = Kinect.kr(20, 10000, f, "X"));
{Out.kr(a, c)}.play;
{Out.kr(b, d)}.play;
```

This creates two Kinect Unit Generators ("RWrist" and "LWrist") and maps their "Y" and "X" positions, given by the `.kr` outputs, to the amplitude and frequency busses respectively. Notice how the desired bounds of each control input are also specified: amplitude being `0-0.5` and frequency being `20-10000` Hertz. A full example of this code is available in the `examples` folder of footnote 1 Page 2. The full list of joint names which are possible are found in the OpenPose documentation [3] .

10. CONCLUSION

In conclusion, SCKinect provides a direct interface to the Kinect in SuperCollider. This allows people to turn their whole body into an immersive controller. Future goals of this plugin are to support other GPU brands and non-GPU based algorithms for performing pose estimation. In addition, a workflow for outputting a "Z" coordinate for each joint, along with the "X" and "Y" coordinates, will be developed. Contributions and feedback are welcome at the repository listed in footnote 1 on Page 2.

11. REFERENCES

[1] "Fastest-selling gaming peripheral," accessed Mar. 01, 2025. [Online]. Available: https://www.guinnessworldrecords.com/world-records/fastest-selling-gaming-peripheral.html

[2] J. Boehm, "NATURAL USER INTERFACE SENSORS FOR HUMAN BODY MEASUREMENT," *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. XXXIX-B3, pp. 531–536, Aug. 2012, accessed Mar. 01, 2025. [Online]. Available: https://isprs-archives.copernicus.org/articles/XXXIX-B3/531/2012/

[3] J. A. De Guzman, K. Thilakarathna, and A. Seneviratne, "Security and Privacy Approaches in Mixed Reality: A Literature Survey," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–37, Nov. 2020, accessed Mar. 01, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3359626

[4] J. Peters, "Microsoft kills Kinect again," Aug. 2023, accessed Mar. 01, 2025. [Online]. Available: https://www.theverge.com/2023/8/21/23840327/microsoft-azure-kinect-developer-kit-discontinued

[5] D. A. Norman, "Natural user interfaces are not natural," *interactions*, vol. 17, no. 3, pp. 6–10, May 2010, accessed Mar. 01, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/1744161.1744163

[6] R. J. Jacob, A. Girouard, L. M. Hirshfield, M. S. Horn, O. Shaer, E. T. Solovey, and J. Zigelbaum, "Reality-based interaction: a framework for post-WIMP interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Florence Italy: ACM, Apr. 2008, pp. 201–210, accessed Mar. 01, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/1357054.1357089

[7] A. Baak, M. Müller, G. Bharaj, H.-P. Seidel, and C. Theobalt, "A Data-Driven Approach for Real-Time Full Body Pose Reconstruction from a Depth Camera," in *Consumer Depth Cameras for Computer Vision*, A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, Eds. London: Springer London, 2013, pp. 71–98, series Title: Advances in Computer Vision and Pattern Recognition. accessed Mar. 02, 2025. [Online]. Available: https://link.springer.com/10.1007/978-1-4471-4640-7_5

[8] Lingzhu Xiang, F. Echtler, C. Kerl, T. Wiedemeyer, Lars, Hanyazou, R. Gordon, F. Facioni, Laborer2008, R. Wareham, M. Goldhoorn, Alberth, Gaborpapp, S. Fuchs, Jmtatsch, J. Blake, Federico, H. Jungkurth, Y. Mingze, Vinouz, D. Coleman, B. Burns, R. Rawat, S. Mokhov, P. Reynolds, P.E. Viau, M. Fraissinet-Tachet, Ludique, J. Billingham, and Alistair, "libfreenect2: Release 0.2," Apr. 2016, accessed Mar. 03, 2025. [Online]. Available: https://zenodo.org/record/50641

[9] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019, accessed Mar. 03, 2025.

[10] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand Keypoint Detection in Single Images using Multiview Bootstrapping," in *CVPR*, 2017, accessed Mar. 03, 2025.

[11] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," in *CVPR*, 2017, accessed Mar. 03, 2025.

[12] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016, accessed Mar. 03, 2025.

[3] https://cmu-perceptual-computing-lab.github.io/openpose/