Skweezee for Processing: A Software Library to Make Squeeze Interactions

Bert Vandenberghe
Kathrin Gerling
Luc Geurts
Vero Vanden Abeele
KU Leuven
Leuven, Belgium
firstname.lastname@kuleuven.be

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
TEI '19, March 17–20, 2019, Tempe, AZ, USA.
Copyright is held by the author/owner(s).
ACM ISBN 978-1-4503-6196-5/19/03.
http://dx.doi.org/10.1145/3294109.3300984

Abstract
Squeeze interaction, defined as squeezing a soft object that affects computed feedback, is a promising interaction technique due to its expressive character. *Skweezee for Processing* is a Processing library that allows makers to implement squeeze interactions in a lightweight manner. The library offers a set of features based on data extraction algorithms and aims to preserve the dynamics of squeezes in the mapping from user action to system output. Distributing *Skweezee for Processing* as an open source library, we invite the community to further investigate the potential of squeeze interactions and to contribute to the extension and improvement of the library. We emphasize the potential for rich squeeze interaction by demonstrating a game that implements squeeze interaction as core mechanic. Additionally, we demonstrate the ease of implementing squeeze interactions in a variety of settings using the *Skweezee for Processing* library.

CCS Concepts
•Human-centered computing → Human computer interaction (HCI); *Interaction devices;*

Author Keywords
Skweezee; squeeze interaction; tangible interaction; open source library; toolkit; making.

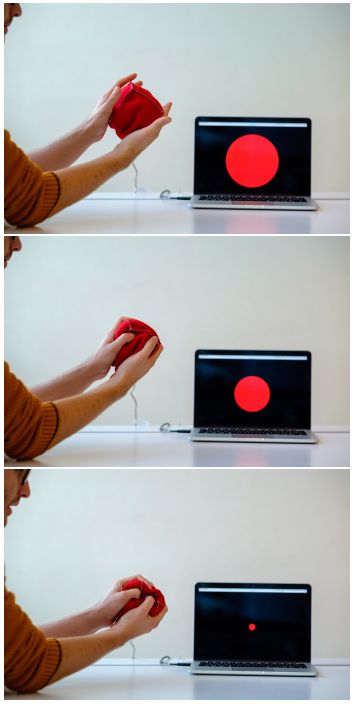

Figure 1: Skweezee for Processing facilitates a quick implementation of squeeze interaction. By generating output using continuous squeeze parameters, the dynamics of squeezes can be preserved from user action to output.

Introduction

Squeeze interaction, defined as squeezing a soft object that affects computed feedback, is a promising interaction technique. The potential of squeeze interaction is discussed in numerous publications in HCI literature. For example, it is presented as “intuitive” [6] and offering “a more engaging experience” [1]. This physical engagement of squeeze interactions finds its use in game controllers, as demonstrated in several projects [1, 6]. Due to the expressive nature [2] and “nuanced control” [4], we also find numerous creative tools implementing squeeze interaction. Examples include musical instruments [9, 11] and sculpting tools [4, 7]. Stienstra & Marti also demonstrate the richness of squeeze interaction by addressing the affective component of squeezes to call a robot [8]. A gentle squeeze elicits a friendly reaction from the robot, while a hard squeeze results in an abrupt reaction. Here, the dynamics of the squeeze have an influence on the reaction of the robot.

In contrast to detailed application case studies, we only find fragmented documentation of the actual technical implementation of squeeze interaction in HCI literature; it appears that there is a frequent disconnect between the measurement of squeezes, signal processing, and feedback generation. For example, sensor resolution only tells part of the story of how user action (the squeeze) and system feedback are coupled. However, the specifics of this coupling, as theoretically presented by authors like Wensveen et al. [12], have a fundamental impact on the interaction itself. We argue that as a research community, we cannot discuss squeeze interaction and its potential in full unless we also discuss the technical implementation in detail. We reinforce this argument by demonstrating the range of technical possibilities offered by our library at TEI, as well as by publishing it as an open source software library to be used by other researchers and designers.

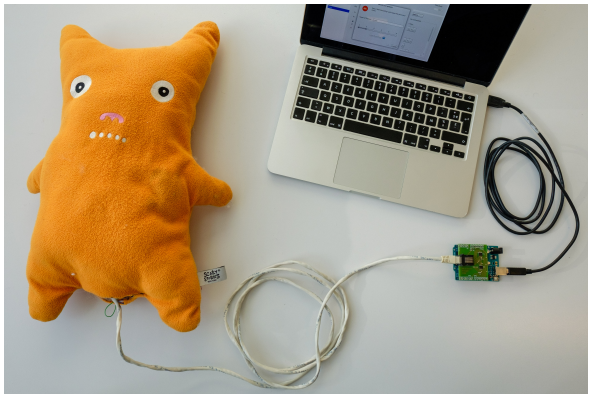

Figure 2: The Skweezee system: a soft squeezable object, an Arduino with custom shield, and desktop software.

Skweezee

The present work is a continuation of the Skweezee system [10] and Skweezee DIY kit [5]. In these preceding projects, a DIY kit was built that allows the use of soft objects such as soft toys for computer input, using conductive wool as stuffing (see Figure 2). An Arduino with a custom shield continuously measures the resistance between 8 electrodes (i.e., 28 possible pairs of electrodes), and sends these 28 values to a computer. Desktop software implementing a support vector machine (SVM) allows the end-user to distinguish multiple gestures, map these gestures to virtual key strokes, enabling the end-user to control a wide range of 3rd party software like flash games or a media player. The DIY kit including the software components was published online in 2015 with the goal of enabling occupational therapists to create interactive therapeutic tools.

While the original Skweezee implementation worked well, we did identify some tensions regarding the use of machine

learning (applying an SVM to recognize gestures). The calculations come with a computational load, as well as the requirement to train the algorithm by repeating the same gesture multiple times before the Skweezee system can be used. In a DIY setting, the implementation also raised a technical barrier for makers. Moreover, by discretizing squeeze input to a limited set of gestures, the continuous character and dynamics of squeeze interaction got lost. As a result, squeeze interaction as implemented by the original Skweezee kit could mainly be used to control applications that are based on discrete user input (such as mouse clicks or key strokes).

Skweezee for Processing Library

In an attempt to preserve the dynamics of squeezes (see Figure 1), the present library processes raw sensor values through data extraction algorithms. Furthermore, we removed the machine learning component from the Skweezee system, replacing it with a lightweight alternative while still facilitating gesture recognition. Finally, for a novice programmer, the Skweezee for Processing library hides technical detail such as setting up a serial connection with an Arduino or implementing signal processing algorithms. As a result, the revised Skweezee software simplifies the process of experimenting with the potential of squeeze interaction that emerges from the richness of a squeeze. However, as the library is published as an open source software project, experienced programmers can still access and adapt those aspects if desired.

The remaining components of Skweezee are not altered: Eight electrodes are placed in a physical object that is filled with conductive wool. The electrodes are connected to an Arduino through the custom shield, which continuously measures the resistance between all possible combinations of the eight electrodes (resulting in a set of 28 values). The data is written to the serial port, for the computer software to retrieve. The Arduino runs at a frequency of 33Hz, meaning that every 30ms a new measurement is available.

Skweezee.measurement()	raw data, array of resistances between all possible pairs of electrodes
Skweezee.vector()	a vector, constructed from the array elements of the measurement
Skweezee.subvectors()	set subvectors, one for each electrode
Skweezee.magnitude()	magnitude of the vector
Skweezee.subdirections()	set of unit vectors per subvector
Skweezee.magnitude(Signal.AVG)	with this parameter, the moving average of the signal is calculated
Skweezee.magnitude(Signal.AVG, Window.EXP)	with this second parameter, the calculation method can be altered
Skweezee.magnitude(Signal.STDEV)	returns the moving standard deviation of the signal
Skweezee.magnitude(Signal.MAX)	the maximum observed value
Skweezee.magnitude(Signal.DIFF)	the first derivative of the signal
Skweezee.magnitude(Signal.DIFF, Stencil.FIVE)	with this second parameter, the derivation method can be altered

Table 1: Examples of available features in the Skweezee for Processing library, for the full set of features and details we refer to the actual library and included documentation.

The library offers basic manipulations, like calculating the average value of all retrieved values in one measurement. One measurement of the Skweezee Arduino shield contains 28 values—one for each possible combination between 8 electrodes. The *average()* parameter thus represents the momentary squeeze in one number. Regarding a measurement as a 28-dimensional vector, we calculate its unit vector which we then use for gesture recognition. To complement these momentary values, the mean value takes into account historic values, filtering noise from the signal at a cost of an introduced latency. Table 1 gives a full overview of the current features of the Skweezee for Processing library.

The end-user imports the library, and can directly access a range of squeeze parameters in the Processing environment. Serial connection and signal processing is handled in the background, as illustrated by the (working) example code below:

```
import processing.serial.*;
import net.skweezee.processing.*;

void setup() {
    Skweezee.connect(this);
}

void draw() {
    float sq = Skweezee.magnitude();
    println(sq);
}
```

As the initial Skweezee implementation was released as a DIY kit online, we prefer to preserve this openness. We therefore release the Skweezee for Processing library and accompanying documentation as an open source project, available at Github[1]. The library is a work-in-progress, allowing us to improve and extend the library over time, based on our own future work and perhaps via contributions by the community. We find endorsement for this approach in recent work by Boucher et al., who distributed their research as an open source project, “a novel approach to disseminating a research methodology” [3].

How to Make Squeeze Interactions with Skweezee

The presented Skweezee for Processing library is part of the Skweezee kit, for the creation of comprehensive interactive systems that implement squeeze interaction. Together with the library, documentation is provided to create a Skweezee from scratch.

We distinguish three main components in a Skweezee system: a soft physical object that can be squeezed; electronics to sense the deformation; and software that processes the sensed deformation and generates output. For each of these three components, we offer multiple options and the corresponding instructions to create the squeeze interaction.

Soft squeezable object

The maker *activates* an existing object such as a soft toy by replacing its stuffing with the conductive material, or can sew a dedicated object. We offer patterns for basic geometric forms in the instructions. Our experiments show that using hot glue or safety pins proves to work if sewing presents a barrier. The use of safety pins allows for quick modifications and thus facilitates rapid iterations.

[1] http://github.com/skweezee

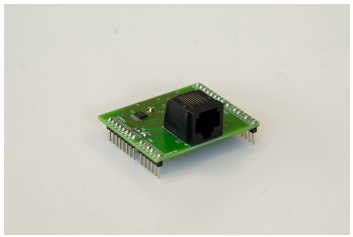

Figure 3: Skweezee sensing shield for Arduino

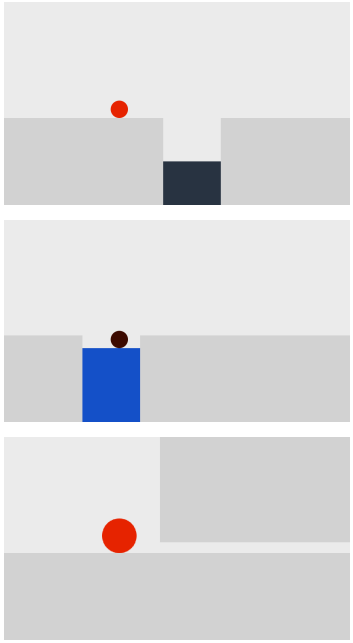

Figure 4: Screenshots of the demo game

Other conductive materials than conductive wool can be used, such as conductive foam used for electronics packaging (black conductive ESD foam), which may be obtained more easily.

Electronics & Arduino
By default an Arduino and our custom shield are used. The shield extends the six available analog pins of a standard Arduino to 28 possible electrode pairs. We use a UTP cable (a standard computer networking cable) to wire the electrodes, as it has 8 wires and is commonly available. The Skweezee for Processing library comes with the required Arduino sketch, and the custom shield schematics are available too.

The library allows a workaround without the shield for a more accessible, though limited implementation. In its most basic configuration, a soft object stuffed with black conductive ESD foam (which is widely available) is used together with an off-the-shelf Arduino Uno. The maker connects two to six wires directly to the analog inputs of the Arduino.

Software
As explained and demonstrated in the Skweezee for Processing section, our library hides technical details from the application, making it easy to implement a squeeze interaction. The Processing environment allows for quick textual, graphical, or even basic auditory feedback. In a set of examples that come with the Skweezee for Processing library, we demonstrate how the Skweezee for Processing library can be applied in different settings.

Demo: Ball Game

In a demonstrator game, the player helps a ball to roll along a route. Various obstacles, such as narrow passages or gaps can be crossed by manipulating game elements using squeezes (see Figure 5).

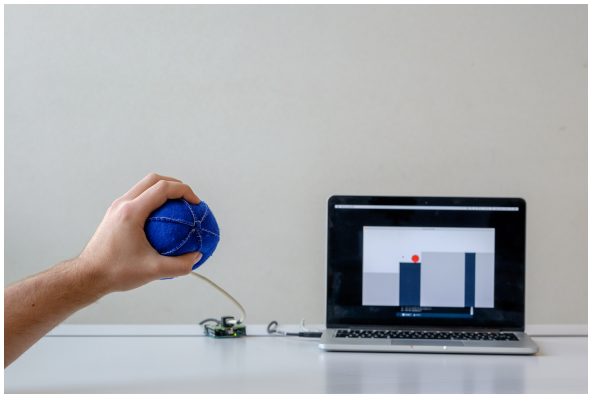

Figure 5: Demo game: manipulate game elements with squeezes.

From a technical perspective, this game demonstrates the ease of how squeeze interaction can be applied in different settings using the Skweezee for Processing library.

Regarding interaction design, the game hints at the potential of squeeze interaction by coupling the squeeze to various properties of elements in the game (such as size or speed). Throughout the game, both the playable character (the red ball) and elements in the environment (such as elevators that help the ball to cross gaps) can be manipulated at different moments in the game.

Conclusion & Future Work

In this paper, we presented a lightweight solution to implement squeeze interaction for makers. The Skweezee for Processing library removes the technical barriers to implement squeeze interaction. By distributing the Skweezee for Processing library as an open source project, we invite the community to build systems integrating squeeze interaction in a variety of application domains and to continue

the investigation of the potential of squeeze interaction. In our own research, we currently apply squeeze interaction in a therapeutic setting using the Skweezee for Processing library.

While Skweezee for Processing requires the use of a computer running Processing, *Skweezee for Arduino* or *Skweezee for Unity* are also explored in our research. These approaches will further extend the range of possibilities for makers and researchers, allowing stand-alone implementations as well as remote or ambient feedback on a variety of platforms. We see the present demonstration as agenda setting for our future research activities.

The research that lead to this Processing library is placing squeeze interaction in the landscape of rich interaction theories. Our research will continue to investigate the characteristics of rich squeeze interaction, and aims to place Skweezee within the grand theories on rich interaction, such as the Interaction Frogger Framework by Wensveen et al. [12]. Furthermore, as we want to explore applications for squeeze interaction beyond the predominant discrete computing paradigm, we will keep extending the set of library features and examples in the future.

References

[1] Tristan Beven, Thuong Hoang, Marcus Carter, and Bernd Ploderer. 2016. HandLog: A Deformable Tangible Device for Continuous Input Through Finger Flexion. In *Proceedings of the 28th Australian Conference on Computer-Human Interaction - OzCHI '16*. ACM Press, New York, New York, USA, 595–604. DOI:http://dx.doi.org/10.1145/3010915.3010933

[2] Harvey Bewley and Anna Vallgårda. 2017. Provoking Performing Objects: Expanding the Design Space for Soft Robotics. In *Proceedings of the 2017 ACM Conference Companion Publication on Designing Interactive Systems - DIS '17 Companion*. ACM Press, New York, New York, USA, 243–248. DOI: http://dx.doi.org/10.1145/3064857.3079154

[3] Andy Boucher, Dean Brown, Liliana Ovalle, Andy Sheen, Mike Vanis, and William Gaver. 2018. TaskCam. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press, New York, New York, USA, 1–4. DOI:http://dx.doi.org/10.1145/3170427.3186536

[4] Sean Follmer, Micah Johnson, Edward Adelson, and Hiroshi Ishii. 2011. deForm: An Interactive Malleable Surface for Capturing 2.5D Arbitrary Objects, Tools and Touch. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*. ACM Press, New York, New York, USA, 527. DOI: http://dx.doi.org/10.1145/2047196.2047265

[5] Argyro Moraiti, Vero Vanden Abeele, Erwin Vanroye, and Luc Geurts. 2015. Empowering Occupational Therapists with a DIY-toolkit for Smart Soft Objects. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction - TEI '14*. ACM Press, New York, New York, USA, 387–394. DOI:http://dx.doi.org/10.1145/2677199.2680598

[6] Vinh P. Nguyen, Sang Ho Yoon, Ansh Verma, and Karthik Ramani. 2014. BendID: Flexible Interface for Localized Deformation Recognition. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '14*. ACM Press, New York, New York, USA, 553–557. DOI:http://dx.doi.org/10.1145/2632048.2636092

[7] Michael Reed. 2009. Prototyping digital clay as an active material. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction - TEI '09*. ACM Press, New York, New York, USA, 339. DOI:http://dx.doi.org/10.1145/1517664.1517733

[8] Jelle Stienstra and Patrizia Marti. 2012. Squeeze me. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction Making Sense Through Design - NordiCHI '12*. ACM Press, New York, New York, USA, 746. DOI: http://dx.doi.org/10.1145/2399016.2399131

[9] Yuta Sugiura, Gota Kakehi, Anusha Withana, Calista Lee, Daisuke Sakamoto, Maki Sugimoto, Masahiko Inami, and Takeo Igarashi. 2011. Detecting shape deformation of soft objects using directional photoreflectivity measurement. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*. ACM Press, New York, New York, USA, 509. DOI: http://dx.doi.org/10.1145/2047196.2047263

[10] Karen Vanderloock, Vero Vanden Abeele, Johan A.K. Suykens, and Luc Geurts. 2013. The skweezee system. In *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*. ACM Press, New York, New York, USA, 521–530. DOI: http://dx.doi.org/10.1145/2501988.2502033

[11] Gili Weinberg, Maggie Orth, and Peter Russo. 2000. The embroidered musical ball. In *CHI '00 extended abstracts on Human factors in computing systems - CHI '00*. ACM Press, New York, New York, USA, 283. DOI:http://dx.doi.org/10.1145/633292.633457

[12] S. A. G. Wensveen, J. P. Djajadiningrat, and C. J. Overbeeke. 2004. Interaction frogger. In *Proceedings of the 2004 conference on Designing interactive systems processes, practices, methods, and techniques - DIS '04*. ACM Press, New York, New York, USA, 177. DOI: http://dx.doi.org/10.1145/1013115.1013140