Design of a Graphical Multi-Input Surface Editor for Mid-Air Haptic Parameter Control

James Weber

This thesis is submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science with Honors.

Barrett, the Honors College

Arizona State University

April 12, 2024

Advisor: Dr. Hasti Seifi, Assistant Professor,

School of Computing and Augmented Intelligence

Second Committee Member: Dr. Ryan Meuth, Associate Professor,

School of Computing and Augmented Intelligence

Abstract

Mid-air haptic technologies enhance user interaction by projecting tactile feedback directly onto a user's skin without requiring physical contact with a feedback device. This thesis details the research and development of a graphical editor which designs 2D curves and 3D surfaces for mid-air haptic parameter control. The editor enables developers working on applications that interface with mid-air haptic technologies to graphically model the relationship between one or two numeric parameters and a modulating phased array output—namely amplitude, amplitude modulation frequency, or drawing frequency. The editor leverages interaction models tailored to the purposes of phased array parameter control, and incorporates just noticeable differences to give developers visual information about end user haptic perception.

I. INTRODUCTION

Ultrasonic phased arrays generate mid-air haptic feedback by leveraging the tactile sensation from acoustic radiation pressure. Through wave field synthesis, arrays of ultrasonic transducers coordinate transducer outputs to produce highly pressurized focal points. These focal points can be rapidly shifted through temporal space, allowing mid-air haptics produced by ultrasonic phased arrays to enhance user interaction across various mediums [1, 2, 3].

Development involving mid-air haptic feedback often involves the need to define a relationship between software parameters and a phased array output parameter. Software parameters are highly case specific, and relate to the software's medium and purpose. In an automotive feedback system, developers may wish to increase phased array emission intensity

when vehicle noise, vibration, or harshness increases. This can require a function which takes as input vehicle velocity, acceleration, or some other variable relating to vehicle roughness, and outputs an ideal amplitude, which directly corresponds with user-perceived intensity. An XR simulation may increase immersion by providing tactile feedback when a user character takes damage. A function can be defined which takes input parameters of user character health or damage taken, and outputs an ideal emission for a relevant phased array output parameter.

The naive approach for designing a relationship between software inputs and a phased array output involves text-based function crafting. This method is insufficient for many phased array software development projects. Text-based function crafting can impede workflow by creating a slow, clunky development cycle with difficult modifiability and hidden errors which would be intuitively recognizable with an effective graphical interface.

This paper details the research and development of a graphical editor which designs 2D curves and 3D surfaces for the purpose of mid-air haptic parameter control. The editor is designed to be used by developers working on software applications which incorporate phased arrays to provide mid-air haptic feedback for end users. The editor gives these developers a graphical interface for intuitive function crafting. The 2D curve editor can be used to define the relationship between one input and one output parameter, and the 3D surface editor can define the relationship between two inputs and one output parameter. The editor takes as input developer-defined software parameters, and outputs an ideal emission for a relevant phased array parameter. The editor also incorporates just noticeable difference (JND) information for the phased array parameters amplitude, amplitude modulation (AM) frequency, and drawing frequency. For these parameters, the developer has the option to receive visual guidance on how the end user may perceive different outputs.

The editor's code repository can be found on https://github.com/jamesweber7/Phased-Array-Parameter-Editor. Section II of this paper describes related works which were used for research during the development of the editor. Section III details the development of the 2D curve editor. Section IV describes the development process of the 3D surface editor. Section V details how JNDs are implemented within the editor. Section VI discusses how the editor can be used as a component by external softwares. Section VII summarizes the editor's final result. Section VIII evaluates future work which could be done to improve the editor.

II. RELATED WORKS

Much research has been done on the coordination of acoustic radiation pressure and wave field synthesis through phased arrays for the production of focal points for mid-air haptic feedback [1, 2, 3]. Uses for phased arrays are being explored in interactive displays [4, 5], automotive feedback systems [5, 6], virtual and augmented reality [5], neuroscience research [5], and many other fields [5].

Implementing research on just noticeable differences in mid-air haptics has been an important piece of this project. Much research has been done on the perception of mid-air haptic feedback [7, 8, 9, 10]. The focus of this project is user perception of single phased array output modulation. Specifically, this project focuses on user perception for modulation of amplitude, amplitude modulation frequency, and drawing frequency [9, 11, 12, 13].

The research and development in this project used many models for defining curves and surfaces. A Bézier curve is a parametric curve that can form a variety of shapes defined by a finite set of endpoints [14, 15]. A composite Bézier curve is created by chaining Bézier curves

together from endpoint to endpoint and ensuring C1-continuity [16]. Other dynamic parametric curves include B-splines [17], β-splines [17], and NURBS curves [18].

The editor was developed using JavaScript. The graphics were created using the p5.js library. The p5.js library uses WebGL for 3D graphics [19, 20, 21].

III. 2D CURVE EDITOR

Development of the editor began with the design of a two-dimensional curve editor. This curve editor serves the purpose of mapping the relationship between a single developer-defined input parameter and a developer-specified output parameter directly tied to phased array emission. The output parameter is assumed to be either AM frequency, drawing frequency, or amplitude. The user can use the editor for another output parameter, but will not receive JND information corresponding to it.

The input parameter can be any developer provided numeric input variable which shares a relationship with the desired phased array output. The 2D curve editor is used whether the developer has one or two input parameters. When the developer creates a surface using two input parameters, the 2D curve editor is still used to define relationships between each individual input parameter and the universal output parameter. This gets explored more in the 3D Surface Editor section of this paper.

Despite its relationship with the 3D surface editor, the 2D curve editor can be thought of as separate from it, and as a compartmentalized component of the overall project. The 2D curve editor is generally agnostic to its role in the 3D surface editor; the primary responsibility of the 2D curve editor is defining the relationship between a single user-defined input parameter and the phased array output parameter. This compartmentalized responsibility gives the developer a

more intuitive starting point of initially working with a 2D relationship before going on to craft a 3D surface.

The 2D curve editor uses a C1-continuous quadratic composite Bézier curve monotonic in the x-direction. Quadratic Bézier curves use three control points [15]. A Bézier curve starts at the first control point and ends at the final control point. These will be referred to as endpoints. Between its endpoints, a quadratic Bézier curve bends towards its central control point. Bézier curves are effective for graphical user interaction because users can intuitively shape Bézier curves by positioning their control points. Composite Bézier curves are formed by chaining one or more Bézier curves together, such that, excluding the first and last Bézier curves, each Bézier curve's final endpoint coincides with the initial endpoint of the Bézier curve after it [16]. They also have the property that each of these coinciding endpoints are smooth, with a continuity of C1 or higher [16]. C1-continuity indicates that the endpoints are linked and share a common tangent direction, ensuring smoothness [16, 22]. This means that the first derivative of two adjacent curves, with respect to the composite curve's parameter, are equal at their shared endpoint [16, 22]. The smoothness provided by C1-continuity is important for applications which require a smooth relationship between their respective input and output parameters. The editor preserves C1-continuity by keeping two connected Bézier curves' join point and respective adjacent control points on the same tangent.

C0-continuity would allow Bézier curve segments to have sharp angles at their join points. C2-continuity would give both curves a common rate of change and shared first and second derivative at their join point [16, 22]. The curve editor does not currently preserve C2-continuity or allow C0-continuity. It could potentially better serve some use cases to give the developer the option to specify continuity order in the future.

Use of the editor begins with a composite Bézier curve made up of one initial Bézier curve segment. The user is able to add segments by clicking within a segment where they would like it to be split into two, making that point become a shared endpoint connecting the two crafted segments. Control points for the two new Bézier curves are placed so that the composite curve does not change its path. Two connected curves can be merged into one by right clicking an endpoint connecting them. All control points can be repositioned through mouse drag interaction.

The composite Bézier curve is bounded in the x- and y-directions. Control points cannot exceed the defined output range. Endpoints of the composite curve cannot be moved in the x-direction, and their x-values mark the input range for the curve's respective input parameter. The editor maintains C1-continuity by dynamically adjusting control points adjacent to one manipulated through user interaction. The editor preserves monotonicity in the x-direction by bounding each control point to a higher x-value than the one before it.

The Bézier curves segments are quadratic, despite cubic Bézier curves often being used instead [15, 16]. During testing, quadratic Bézier curve segments were more intuitive to use than cubic Bézier curves in most situations. With quadratic Bézier curve segments, the developer only needs to modify one control point besides the endpoints per segment. Because the developer can model more diverse curve shapes by creating more segments, quadratic Bézier curve segments require the developer to modify fewer unnecessary control points while giving the same degree of freedom as would be given with cubic Bézier curve segments.

B-spline, instead of composite Bézier curve, would have been another effective curve type. However, composite Bézier curves allow developers to explicitly define which points the curve should run through by dividing a segment into two and placing their join point at the

position they need the curve to pass through. The fluidity of B-spline curves may better serve purposes where general relationships are prioritized and the developer does not need to define explicit positions along the curve [17].

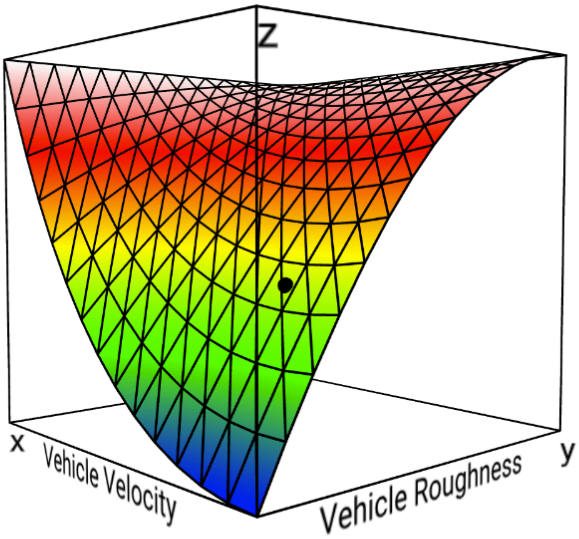
IV. 3D SURFACE EDITOR

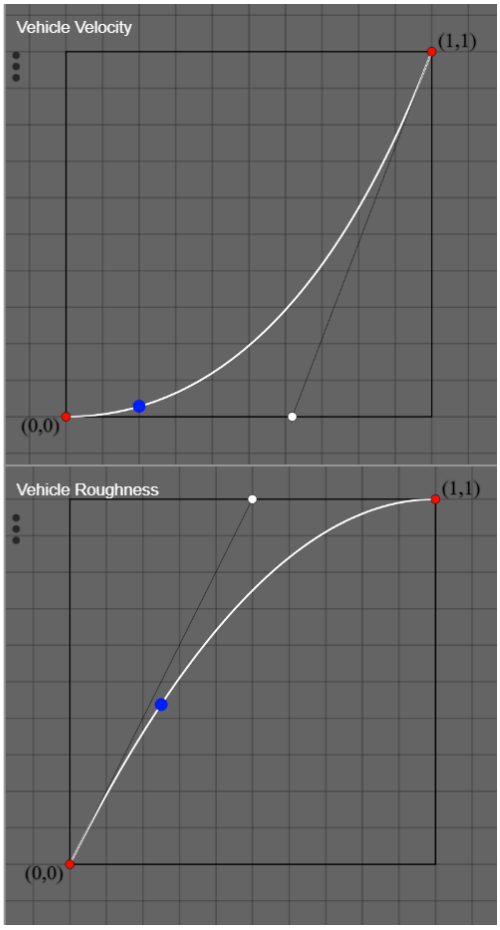
The 3D surface editor maps two developer-defined input parameters to a single developer-specified output parameter. There are many approaches to 3D surface design which were considered during early development. One approach is to use methods from 3D solid sculpting tools to shape a 3D surface with user mouse interaction [23]. This method of interaction gives the user a high level of control over the surface, but can require more user interaction than a simpler solution would need. The precise control given by sculpting is unnecessary for most use cases for this editor. Depending on implementation, sculpting can also require a steep learning curve.

Some 3D surfaces are defined by a matrix of control points, such as with Bézier surfaces or NURBS surfaces [18, 22]. As with 2D Bézier curves, designing these surfaces can be intuitive through interaction with control points. However, because these surfaces are defined by a matrix of control points, designing these surfaces often requires repositioning a significantly large number of control points.

This surface editor defines a surface by interpolating between the curves along its edges. These curves are defined by the 2D curve editor. This configuration allows the developer to think in terms of the 2D relationships between each individual input parameter and the output parameter. Because these interactions relate to each other, the interpolation between these curves is effective for defining a 3D relationship between both input parameters and the output

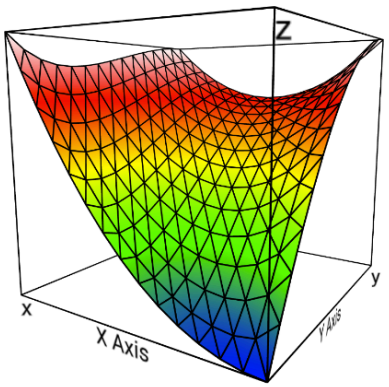
parameter. Curves designed using the 2D curve editor are taken as edges of the surface, placed along the x and y axes. By interpolating between these curves, the editor can take any x and y values from the domain and return a z-value relating to phased array output. The surface editor uses two-edge interpolation functions and four-edge bilinear interpolation Coons patches to blend the curves along its edges.





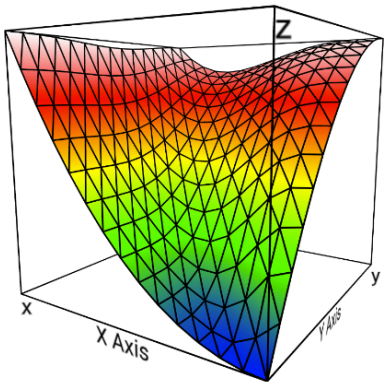
The 3D surface editor was initially created using a linear interpolation function. The surface is defined by each point (x, y) within the domain having an output equal to the weighted

average between the values of its projection onto the curve along the x and y axes. Each axis is weighted by the ratio of its proximity to the input point over the proximity of both axes to the input point. This straightforward method allows for high predictability in surface behavior, and is the default interpolation function.



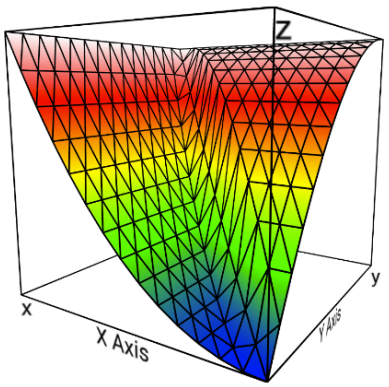
Linear Interpolation

The surface editor can also use other interpolation functions, giving a non-linear weight to each axis based on its proximity to the respective point in the x and y directions. Along the diagonal from the vertex at x=0, y=0 to the terminal point defined by the maximum extents of the domain and range along the x and y axes, where the x and y values are equal, weighted interpolation functions maintain the same z-values as the linear interpolation function.



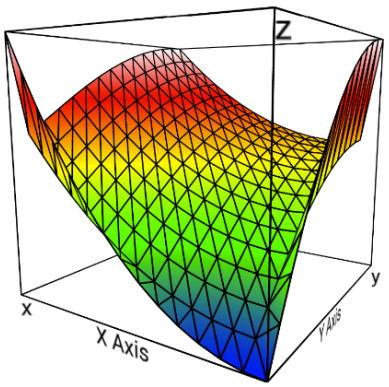
Polynomial Interpolation
degree=3

The surface editor provides the option to use polynomial interpolation, with a positive degree defined by the user. Polynomial interpolation with a degree of 1 is the same as linear interpolation. A degree of less than 1 causes the z-value to snap more quickly to be near the average value before getting close to the central diagonal. A degree greater than 1 will cause the z-value to linger near the z-value of its closer axis for longer, approaching the average value only as it nears the diagonal. A high degree will cause the diagonal to be pronounced, causing a sharp snap from the z-values along the x-axis to the z-values along the y-axis right at the diagonal.



Polynomial Interpolation
degree=100

Other interpolation methods can be defined for the editor. For instance, sinusoidal interpolation can result in a natural-feeling smooth transition from one axis to the other. This transition leverages the sine wave to modulate each



Polynomial Interpolation
degree=0.1

point's weight according to relative proximity to an axis. Sinusoidal interpolation was implemented in the editor, but does not behave significantly differently from polynomial interpolation. Each option presented by the editor is another option developers may need to learn and remember in order to experience perceived control over the editor. Because of this, the inclusion of a sinusoidal interpolation option did more harm than good, and is therefore excluded from the final editor version.

Besides these two-edge interpolation functions, the 3D surface editor makes use of bilinear interpolation through Coons patches. A Coons patch is a surface that takes four edges and blends between them in a natural-feeling way [22, 24]. The Coons patch poses two benefits: it allows up to four edges to affect the total output, and it provides a transition that behaves significantly different from the interpolation functions, thereby giving the user more options to choose from when determining which surface to use for their edges.

Despite the Coons patch requiring four edges as input, the surface editor can make use of the Coons patch with fewer than four edges as input from the user. The editor does this by projecting opposing edges from the two edges the user designs in the curve editor. The user has several options for how these projections can work.

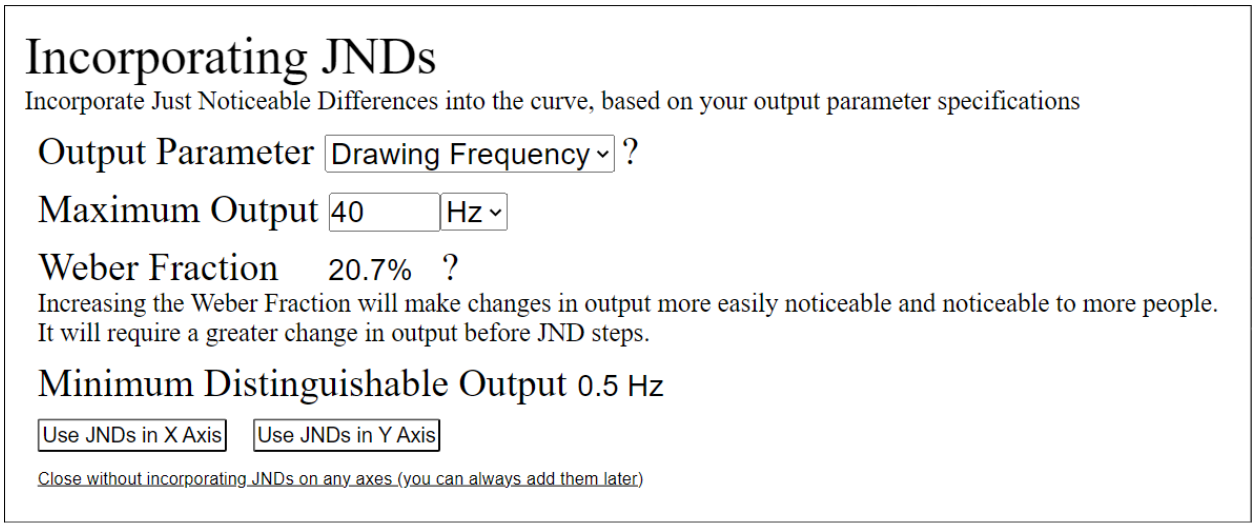
The default projection option is to calculate opposing edges that would be the same as from linear interpolation. This projection is made the standard option because in testing, it tends to give a perceived feeling of naturality for many general use cases. Alternatively, the surface editor can construct opposing edges that form straight edges from the end of the x-axis to the end of the y-axis. Another option is for the x-axis and y-axis to project directly onto their opposing edges. A drawback of this option is that it requires each endpoint for all four edges to have an equal z-value. This makes the option not viable for many use cases, such as when a developer

wishes for the output to ascend or descend as it approaches the extents of the domain. The Coons patch can also use up to four user-defined curves as edges, making it the ideal option when the user desires control over the edges opposing the x and y axes.

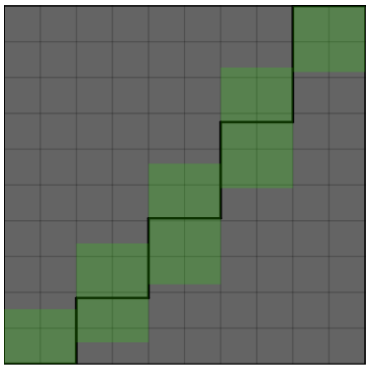
V. JUST NOTICEABLE DIFFERENCES FOR PHASED ARRAY OUTPUT

The 2D curve editor enables use of variable step sizes based on just noticeable differences in order to provide a template for crafting their curve in a way that will translate well to real-world mid-air haptic perception. Each step corresponds to a fixed output level for an input range. These steps are based primarily on two developer-specified parameters: the phased array output parameter and the maximum output supported by the phased array. The supported output parameters are AM frequency, drawing frequency, and amplitude [9, 11, 12, 13]. Each output parameter has a corresponding Weber fraction, which specifies how much an output needs to change for a phased array user to distinguish between two separate outputs. Developers can adjust the Weber fraction after the recommended Weber fraction is given. Increasing the Weber fraction causes steps to be more easily distinguishable, and distinguishable to more people.

A minimum distinguishable output, which defines the minimum output that is distinguishable from no output, is given based on the phased array output parameter. With the maximum output, the editor computes the maximum number of monotonic steps distinguishable by phased array users.



Window For Incorporating Just Noticeable Differences



Steps with Just Noticeable Differences Information

Based on all these parameters, developers can edit steps with feedback on whether the steps are within the defined bounds for just noticeable differences. Developers can choose between ascending or descending steps by default, and the number of steps they wish to have. This will compute a set of steps that maximizes the probability of each step being distinguishable. Developers can manipulate these steps dynamically to design their desired arrangement. Developers can also set these steps as a template for a curve, so they can design their desired curve according to the computed JND information.

VI. USE WITH EXTERNAL SOFTWARE

The editor can be used on its own, but was developed with the intention of being used as a component in external softwares [25, 26]. Developers can have external softwares access the editor's software through a content delivery network (CDN), or by directly integrating the

editor's repository within the external software's code. External softwares can interface with the editor through a static JavaScript class, ParameterEditor.

Calling ParameterEditor's initialization method will connect with the editor through an iframe. An iframe, or inline frame, is an HTML element that embeds an HTML page within another HTML document. An external software can have an existing iframe that ParameterEditor can connect to, or ParameterEditor can create a new iframe. ParameterEditor can then be used to interface with the editor, such as by returning a table of computed output values designed by the surface.

VII. SUMMARY

The final result is a software that can take one or two user-defined input parameters and give the user an interface that effectively defines their relationship with an output parameter. Developers can use the software as a standalone application or as a component in an external development application. The user can edit curves and surfaces, incorporate just noticeable differences, and choose from multiple methods of defining a surface based on their designed curves.

VIII. MOVING FORWARD

Further work can be done in many areas related to this project. There are many types of 2D curves that could have been adapted to the purposes of the 2D editor. Splines specifically have been adapted to many purposes, including B-spline, β-spline, NURBS curves, and

Kochanek–Bartels [17, 18, 27]. Adapting some of these to the purpose of the 2D curve editor and giving developers the option to use them could potentially yield increased productivity.

An option to use cubic or higher-order Béziers, instead of quadratic Béziers, could give developers more freedom when designing curves. Allowing developers to define the order of continuity (C0, C1, C2 or higher) instead of only C1 could also better serve many purposes.

The 3D surface editor could potentially serve more use cases by giving developers more options for surface editing. Bézier or NURBS surfaces, though often tedious to interact with, give an intuitive method for interaction, with more freedom over the center of the surface than the surface editor currently gives developers. Sculpting methods could give developers an even higher degree of freedom. Although these methods were decided against for the general use cases of this editor, many developers could benefit from the greater range of surface shapes allowed by these surfaces.

The two-edge interpolation methods (linear, polynomial, and sinusoidal) all share the same central diagonal where both axes are weighted equally. However, not all input parameters have equal relevance to ideal output. Giving each axis a modifiable weight, thereby allowing an axis to have a larger or smaller effect on the surface, could resolve this.

Additionally, a theoretical feature that could be implemented is a JND visualization map, which would transform a developer's smooth surface into stepped plateaus where just noticeable differences may be perceived. This would help developers visualize where there are perceived increases and decreases by the phased array, and let developers explicitly define where these should be felt within a 3D domain. However, the JNDs would depend on the direction in which the input shifts in the x and y directions along the 3D surface, and would therefore not be entirely representative of JNDs if the developer doesn't use the JND map as their actual surface.

Nonetheless, exploring this theoretical feature could yield information relevant to developers, and give them more explicit control over where JNDs can be expected over the domain.

More controls could be added to the interface with external softwares. Methods to serialize and deserialize the editor's state would be necessary for use of the editor over multiple sessions. It could also be useful to give external softwares methods to convey input and output parameter information.

Finally, more research could go into JND information the curve editor uses. Some JND parameters involve speculation, and all could benefit from further testing.

IX. REFERENCES

[1] T. Hoshi, M. Takahashi, T. Iwamoto, and H. Shinoda, "Noncontact Tactile Display Based on Radiation Pressure of Airborne Ultrasound," in *IEEE Trans. Haptics*, 2010, vol. 3, no. 3, pp. 155-165, doi: 10.1109/TOH.2010.4.

[2] T. Carter, S. A. Seah, B. Long, B. Drinkwater, and S. Subramanian, "UltraHaptics: Multi-Point Mid-Air Haptic Feedback for Touch Surfaces," in *Proc. 26th Annu. ACM Symp. User Interface Softw. and Technol.*, ACM, 2013, pp. 504-513, doi: 10.1145/2501988.2502018.

[3] T. Iwamoto, M. Tatezono, and H. Shinoda, "Non-contact Method for Producing Tactile Sensation Using Airborne Ultrasound," in *Haptics: Perception, Devices and Scenarios. EuroHaptics 2008*, M. Ferre, Ed., vol. 5024, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 504-513, doi: 10.1007/978-3-540-69057-3_64.

[4] H. Limerick, R. Hayden, D. Beattie, O. Georgiou, and J. Müller, "User engagement for mid-air haptic interactions with digital signage," in *Proc. 8th ACM Int. Symp. Pervasive Displays (PerDis '19)*, Palermo, Italy, Jun. 2019, pp. 1-7, doi: 10.1145/3321335.3324944.

[5] O. Georgiou, W. Frier, and O. Schneider, "User Experience and Mid-Air Haptics: Applications, Methods, and Challenges," in *Ultrasound Mid-Air Haptics for Touchless Interfaces*, O. Georgiou, W. Frier, E. Freeman, C. Pacchierotti, and T. Hoshi, Eds. Cham, Switzerland: Springer, 2022, ch. 2, doi: 10.1007/978-3-031-04043-6_2.

[6] K. Harrington, D. Large, G. Burnett, and O. Georgiou, "Exploring the Use of Mid-Air Ultrasonic Feedback to Enhance Automotive User Interfaces," in *Proc. 10th Int. ACM Conf. Automot. User Interfaces and Interactive Veh. Appl. (AutomotiveUI '18)*, Toronto, Canada, Sept. 2018, pp. 11-20, doi: 10.1145/3239060.3239089.

[7] D. Pittera, O. Georgiou, and W. Frier, "'I See Where This Is Going': A Psychophysical Study of Directional Mid-Air Haptics and Apparent Tactile Motion," *IEEE Trans. Haptics*, 2023, vol. 16, no. 2, pp. 1-12, doi: 10.1109/TOH.2023.3280263.

[8] D. Hajas, D. Pittera, A. Nasce, O. Georgiou, and M. Obrist, "Mid-air haptic rendering of 2D geometric shapes with a dynamic tactile pointer," *IEEE Trans. Haptics*, 2020, vol. 13, no. 4, pp. 806-817, doi: 10.1109/TOH.2020.2966445.

[9] T. Howard, G. Gallagher, A. Lécuyer, C. Pacchierotti, and M. Marchal, "Investigating the Recognition of Local Shapes Using Mid-air Ultrasound Haptics," in 2*019 IEEE World Haptics Conf. (WHC)*, Tokyo, Japan, pp. 503-508, doi: 10.1109/WHC.2019.8816127.

[10] R. Takahashi, K. Hasegawa, and H. Shinoda, "Tactile stimulation by repetitive lateral movement of midair ultrasound focus," *IEEE Trans. Haptic*s, 2019, vol. 13, no. 2, pp. 334–342.

[11] C. Lim, G. Park, and H. Seifi, "Designing Distinguishable Mid-Air Ultrasound Tactons with Temporal Parameters," in *Proc. CHI Conf. Human Factors in Comput. Syst. (CHI '24)*, Honolulu, HI, USA, May 2024, doi: 10.1145/3613904.3642522.

[12] K. Wojna, O. Georgiou, D. Beattie, W. Frier, M. Wright, and C. Lutteroth, "An Exploration of Just Noticeable Differences in Mid-Air Haptics," in *2023 IEEE World Haptics Conf. (WHC)*, Delft, The Netherlands, pp. 410-416, doi: 10.1109/WHC56415.2023.10224388.

[13] I. Rutten, W. Frier, and D. Geerts, "Discriminating Between Intensities and Velocities of Mid-Air Haptic Patterns," in *Proc. Haptics: Sci., Technol., Appl.: 12th Int. Conf.*, EuroHaptics 2020, Leiden, The Netherlands, Sep. 2020, pp. 78-86, doi: 10.1007/978-3-030-58147-3_9.

[14] L. Qiao, W. Ding, X. Qiu, and C. Zhang, "End-to-End Vectorized HD-map Construction with Piecewise Bézier Curve," in *2023 IEEE/CVF Conf. Comput. Vision and Pattern Recognit. (CVPR)*, IEEE, pp. 13218-13228, doi: 10.1109/CVPR52729.2023.01270.

[15] M. E. Mortenson, *Mathematics for Computer Graphics Appl.*, 2nd ed. New York, NY, USA, Industrial Press, 1999.

[16] E. V. Shikin and A. I. Plis, *Handbook on Splines for the User*. Boca Raton, FL, USA, CRC Press, 1995.

[17] T. N. T. Goodman, "Properties of β-splines," *Journal of Approx. Theory*, 1985, vol. 44, no. 2, pp. 132-153, doi: 10.1016/0021-9045(85)90076-0.

[18] L. Piegl and W. Tiller, *The NURBS Book*, 2nd ed. Berlin, Germany, Springer, 1997, doi: 10.1007/978-3-642-59223-2.

[19] "JavaScript Reference," Mozilla Developer Network. Accessed: Apr. 8, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

[20] p5.js Community, "p5.js Reference," Accessed: Apr. 8, 2024. [Online]. Available: https://p5js.org/reference

[21] "WebGL API," Mozilla Developer Network. Accessed: Apr. 8, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

[22] G. Farin, *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*, 3rd ed., San Diego, CA, USA, Academic Press, Inc., 1992.

[23] T. A. Galyean and J. F. Hughes, "Sculpting: An Interactive Volumetric Modeling Technique," in *Proc. 18th Annu. Conf. Comput. Graph. and Interactive Techniques*, SIGGRAPH '91, ACM, 1991, pp. 267-274, doi: 10.1145/122718.122747.

[24] S. A. Coons, "Surfaces for Computer-Aided Design of Space Forms," MIT Lincoln Laboratory, Technical Report MIT-LCS-TR-41, Jun. 1967. Accessed: Mar. 20, 2024. [Online]. Available: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-041.pdf

[25] K. John, Y. Li, and H. Seifi, "AdapTics: A Toolkit for Creative Design and Integration of Real-Time Adaptive Mid-Air Ultrasound Tactons," in *Proc. CHI Conf. Human Factors in Comput. Syst. (CHI '24)*, Honolulu, HI, USA, May 2024, doi: 10.1145/3613904.3642090.

[26] H. Seifi, S. Chew, A. J. Nascè, W. E. Lowther, W. Frier, and K. Hornbæk, "Feellustrator: A Design Tool for Ultrasound Mid-Air Haptics," in *Proc. 2023 CHI Conf. Human Factors in Comput. Syst. (CHI '23)*, New York, NY, USA, Assoc. Comput. Mach., Art. no. 266, pp. 1–16, doi: 10.1145/3544548.3580728.

[27] D. H. U. Kochanek and R. H. Bartels, "Interpolating splines with local tension, continuity, and bias control," in *Proc. 11th Annu. Conf. Comput. Graph. and Interactive Techniques (SIGGRAPH '84)*, New York, NY, USA, 1984, pp. 33-41, doi: 10.1145/800031.808575.