7

Input Processing and Geometric Representations for 3D Sketches

Johann Habakuk Israel, Mayra Donaji Barrera Machuca, Rahul Arora, Philipp Wacker, and Daniel Keefe

Even though some commercially available 3D sketching systems already exist, the need to develop own systems may emerge, for example in the context of research projects, in the context of teaching, or to extend the functionality of existing immersive systems with 3D sketching features. After the unique characteristics of 3D sketching were presented in the previous chapter, this chapter outlines the essential steps for generating a 3D sketch using freehand user input. Many of the principles of 2D sketching can also be applied to 3D and will not be repeated here. Instead, selected methods that have proven themselves in practice in 3D are explained.

As immersive 3D sketching systems are highly interactive, all steps must be performed in real time and post-processing approaches are not suitable here. Besides the hardware requirements described in Section 6.1, the following steps must be taken into account on the software side when realising a freehand 3D sketching system:

tracking: recording the position, orientation, and states of the buttons of the interaction devices used (see Section 6.1.2),

filtering: filtering of the input data, for example, smoothing,

sampling: transfer of the input data into an internal data structure, if necessary removal of unnecessary supporting points,

mesh creation: creation of a representable mesh based on the supporting points, for example by extrusion of a basic shape (brush), and

rendering: real-time representation of the mesh

Immersive sketching systems typically provide additional features such as deleting, editing, saving, and loading sketch entities. Since these are not significantly different from 2D sketching, they are not discussed here. In addition, immersive sketching systems typically have 3D menus for selecting and parameterising specific sketching functions such as brush, colour, and curve properties, among others. An overview of the design of 3D menus is provided by Dachselt and Hübner [1] and LaViola et al. [2]. Further user interface-related guidance on processing user input to control immersive 3D sketching systems can be found in Jackson and Keefe [3].

7.1 Filtering

As mentioned in Section 6.1.2, the input devices used in 3D sketching usually provide data in the full six spatial degrees of freedom. Many methods for generating stroke geometry take the orientation into account, for example, in the creation of calligraphic sketches. Here the orientation of the interaction device has an important influence on the resulting geometry. The filter methods described here only refer to translational data. Filter methods also exist for rotational data, but are much more sophisticated. Hartley et al. [4] provide a good overview of corresponding methods. Some 3D frameworks also offer the possibility to interpolate rotations using quaternions.

In freehand 3D sketching, input data is typically filtered for two reasons, namely to reduce or smooth out hardware-induced noise, typically caused by the inaccuracy of the tracking systems used, and to reduce small jittery movements caused by the user when guiding the 3D input devices.

Smoothed user input produces much more aesthetic and satisfying sketches than unfiltered ones. At the same time, filtering causes slight delays, which add to the delays already caused by the tracking and rendering system. This is especially noticeable with fast sketch movements, such as when drawing wavy lines quickly. In AR and projection-based environments such as CAVEs and powerwalls, users typically see the physical input devices directly. Here, the delay is visible in such a way that during sketch movements, the extrusion point of the virtual ink follows the position of the physical input device or the physical pen tip at a noticeable distance. In HMD-based systems, where users hold the interaction devices in their hands but typically only see them as rendered 3D geometries in the virtual environment, pen tip and extrusion point are always synchronous. On the other hand, there can be a disturbing offset between the position of the virtual input device and the position of the physical input device held in the hand,

which users perceive although they cannot see the physical input device directly, but feel where and how they are holding it [visual-proprioceptive conflict, cf. 5].

There are several filtering methods that differ in particular in the degree of smoothing and delay of user input[1]. The easiest way to filter user input is the Moving Average Filter. This filter uses a certain number of the last positions X_n measured by the interaction device and forms a simple average value $\bar{X}_n$. Although good smoothing can be achieved with this filter, the delay increases noticeably with the number of positions considered. In addition, sudden changes in direction of the pen guidance cannot be satisfactorily represented by this filter. To address this problem, the Exponential Moving Average Filter, given by Equation 7.1, can be used. This filter calculates the filtered value $\bar{X}_n$ using a weighted average of the current position X_n of the interaction device and the last filtered value $\bar{X}_{n-1}$. The higher the weighting factor a weights the current position, the higher the responsiveness to motion changes, but with a lower smoothing effect.

$$\bar{X}_n = aX_n + (1-a)\bar{X}_{n-1} \tag{7.1}$$

This filter reacts better to changes in direction, but still generates noticeable delays when smoothing is strong. Double Exponential Smoothing Filters, defined by Equation 7.2 and Equation 7.3, offer even better response to rapidly changing directions of motion [6]. They include the trend (b_n) of the movement, which is determined by an average of the difference between the last two filter results ($\bar{X}_n$, $\bar{X}_{n-1}$) and the last calculated trend, weighted with the factor γ.

$$b_n = \gamma(\bar{X}_n - \bar{X}_{n-1}) + (1-\gamma)b_{n-1} \tag{7.2}$$

$$\bar{X}_n = aX_n + (1-a)(\bar{X}_{n-1} + b_{n-1}) \tag{7.3}$$

γ factors closer to 1 are appropriate for reliable tracking systems with less noise and result in shorter delays. For less accurate tracking systems, such as magnetic tracking systems, γ factors lower than 0.1 may be necessary to achieve smooth lines, resulting in noticeable spatial lag. The delay becomes less noticeable if the tracking system provides a high updated rate, that is, many samples per second. Reading the tracking system with the highest possible frequency is, therefore, especially helpful if the tracking data is

[1] For immediate feedback it is possible to use a temporary point (typically X_n) and replace it with the filtered point ($\bar{X}_n$) when the filtering is done.

noisy. Usually this requires a process that is independent of the rendering thread, which runs at a higher frequency and writes the tracking data into a queue. As mentioned in Section 6.2.2.2, research by Keefe et al. [7], Arora et al. [8], Barrera Machuca and Stuerzlinger [9], and Batmaz et al. [10] indicates that freehand sketches are often noisier in depth than in horizontal and vertical directions. Therefore, it may be beneficial to smooth tracking data more strongly in the egocentric depth direction.

Another useful method to improve the visual quality of sketches was proposed by Thiel et al. [11]. To distinguish details intended by the drawer from noise caused by the tracking system, they use the speed at which the line was drawn. The higher the speed of the drawing movement, the more the user input is smoothed.

7.2 Sampling

As soon as the user presses the button provided on the interaction device, the position and orientation data supplied by the tracking system are recorded to create the sketch geometry. Typically, this data is filtered as described above before it is processed further. The next step in the processing chain is called sampling. This is where it is decided which of the input data will be used to create the geometry. It is important not to use all of the data generated by the tracking system, otherwise a lot of data could be generated that has no visual effect. For example, if the user holds the input device still with the button pressed, approximately identical position data is generated in each scanning run. If all of this data were to be transferred, the result would be many lines in a very small space that would hardly be recognisable to the observer. In addition, a large number of points in a single spot could distort the appearance of the line in a way not desired by the user. At the same time, the data set to be processed (supporting points of the sketch) would be considerably inflated, so that all subsequent processes would be unnecessarily burdened.

In the sampling process, therefore, only those points are considered that are sufficiently far apart to be perceived by the user. The easiest way to do this is to define a minimum distance that points must maintain from each other. For each position supplied by the tracking system, the Euclidean distance to the previous position is then determined. The new point will only be considered if this distance is greater than the minimum distance. Furthermore, it is possible to check whether several consecutive points of a line lie on a straight line. All points to which this applies can be removed. An efficient method for selecting the relevant position data for generating strokes is the

Douglas-Peucker algorithm [12] (refer to Chapter 3 for a brief description of this algorithm).

In some cases, the Nyquist frequency [13] may be exceeded, which should be at least twice the frequency of the signal. In other words, the system does not collect enough position data to correctly capture and accurately reproduce the user's sketching movements. For example, when the user performs very fast zigzag movements, and the created path does not accurately reproduce the user's sketch movements. In this case, too little sampling is often not due to too slow tracking technology but results from an unfavourably implemented processing pipeline. To achieve better results, developers can delegate the retrieval of tracking data to a parallel process.

7.3 Geometric representations

After the user's pen or sketch movements are captured and filtered, they must be stored in an internal data structure of the sketching application. Many 3D frameworks offer specific functionalities for this purpose, for example, in Unity3D a so-called line renderer exists, which creates a geometric representation from a list of points and renders it by connecting the points with straight lines [14]. The disadvantage of using such functionalities is that many dependencies to the used 3D framework arise in the program code and adjustments are more difficult to implement.

The simplest way to do this oneself is to use the scene graph of the 3D framework used. Incoming motion information is then converted directly into representable 3D geometry, typically by extrusion (see below). Since the scene graph typically only stores the geometric representations, but not the original sketch movements, this variant has the clear disadvantage that changes to sketches and the loading and saving of sketches are very difficult. It is, therefore, a good idea to save the (filtered) sketch motion data and use it to generate renderable sketch elements in a separate step. Since in this case the saved sketch entities are the parameters of the generated geometry, this approach can be described as parametric modelling [15]. Parametric modelling is already a standard procedure in CAD. Every CAD tool has a so-called modelling kernel, which realises the processing and manipulation of 3D model data as well as the generation of displayable 3D elements, a process which is called tessellation. In the area of open source software, the modelling core Open Cascade is available [16].

For smaller projects, which only need a few sketch element types, such as freehand strokes, straight lines or simple Bézier surfaces, it is also possible

to develop one's own modelling kernel which should have an interface for generating sketch elements, for manipulating them and for persistence (loading and saving).

Section 8.2.1.1 describes a variety of basic geometric shapes that are possible and conceivable in immersive sketching applications. This section focuses on the generation of strokes, the most basic geometric elements of sketches. In addition, other elements such as surfaces or volumetric objects can be created in a sketching manner. A number of excellent textbooks are available for more detailed illustrations of internal data structures, parameterisation, mesh creation, and rendering [e.g. 17, 18, 19].

7.3.1 Strokes

Strokes are the most important geometric elements of sketches. There are several ways to generate internal data representations from the information about sketch movements and to display them. The simplest possibility is to save the support points generated in the sampling process for each line in a list and to connect them to each other with a cylinder or a straight tube. If the support points are close together, this method can already provide satisfactory results. However, in the case of fast sketch movements which, due to the limited sampling rate of the tracking system, provide support points that are comparatively far apart, visible corners and edgy curve shapes can occur in this method. To avoid this problem, the use of parametric curves is recommended. These describe the shape of the curve using a mathematical function that receives the control points as input values. Along this curve a basic shape is then extruded, so that renderable elements are generated; a process called tessellation.

7.3.1.1 Parametric curves

If the support points of a line resulting from the sampling process are described by $s_0, \cdots, s_n$, where n is the number of support points, then a parametric curve can be created using a function $p(t)$, which receives a subset of the support points as input and calculates the points $p_0, \cdots, p_m$ of a line. The number of support points n and the number of line points m typically differ. This makes it possible to draw lines with higher resolution than the interpolation points would actually provide. However, this is accompanied by the fact that the drawn line can deviate from the real line, since the lines must be synchronous only at the support points. A selection of simple parametric

curves is presented below. A comprehensive overview can be found, for example, in Akenine-Möller et al. [19, chapter 17] .

7.3.1.2 Linear interpolation

The simplest and still, at least in research projects, widely used form of a parametric curve is linear interpolation between two interpolation points. For this purpose, the slope between the two points is calculated. Each point of the curve can then be determined using Equation 7.4.

$$p(t) = s_{i-1} + t(s_i - s_{i-1}), \quad t \in [0,1]. \tag{7.4}$$

The result of the linear interpolation is a piece-wise linear curve, which connects all points by lines. Since no new line points p can be determined by the linear interpolation, which do not already lie on a straight line between $s_i - s_{i-1}$, a parametric description is usually omitted here.

7.3.1.3 Bézier curves

The jagged stroke pattern achieved by linear interpolation is not sufficient in many application contexts. Smooth, aesthetic transitions between the line segments, or, in mathematical terms, a geometric continuity of order G^1 (tangent continuity) or even G^2 (curvature continuity), are often expected. To achieve this, Bézier curves which give G^1 continuity are used in many cases [20] (see Box 2.6 in Section 2.2 for further details on Bézier curves). These curves also use the data from the sampling process as support points. According to the degree k of the Bézier curve, $k+1$ points are used to calculate a curve segment. The resulting curve lies within a convex hull of the control points and runs through the first and last of the control points. The continuous progression is achieved by weighting the influence of the points S_i on the resulting curve differently using the Bernstein polynomial B_i^k according to the parameter t. Equation 7.5 gives an example of a cubic Bézier curve drawn between four points. In computer graphics, cubic Bézier curves with $k = 3$ are most frequently used.

$$p(t) = B_0^3(t)p_0 + B_1^3(t)p_1 + B_2^3(t)p_2 + B_3^3(t)p_4, \quad t \in [0,1]. \tag{7.5}$$

In Bézier curves, the resulting curve is most strongly pulled towards the support points to which it is closest, but without reaching them. While this results in smoothly continuous curves, it is also the biggest problem of using Bézier curves for sketches since the curve shown does not pass through

the curve described by the stylus, except at the first and last point of each curve segment, but only approximates it. This deviation can result in the visual appearance of the sketch not corresponding to the user's intention. The fact that some of the support points are not on the stroke can also be problematic if users want to change their sketch later. For example, if users wanted to change the curve, the easiest way to do this would be to move the support points accordingly. However, if they do not lie directly on the sketched line, these manipulations become unnecessarily difficult. For this reason, representations should be used for the visualisation of sketched lines, where the support points lie directly on the sketched line, which can be achieved by Cubic Hermite interpolation.

7.3.1.4 Cubic Hermite interpolation

The advantage of Cubic Hermite interpolation[2] is that it is relatively easy to control [19]. It is sufficient to define a tangent m_k at each support point p_k. The sketched line can easily be transformed into individual Hermite curve segments. Each support point is then both the end and the beginning of a segment. The tangent of each interpolation point can be calculated by the vectors from the previous p_{k-1} to the current interpolation point p_k and from the current interpolation point p_k to the following interpolation point p_{k+1} as defined in Equation 7.6.

$$m_k = \frac{p_k - p_{k-1}}{2} + \frac{p_{k+1} - p_k}{2} \tag{7.6}$$

The tangents can then be used to interpolate the sketched line using the cubic polynomials in Equation 7.7.

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (-2t^3 + 3t^2)p_1 + (t^3 - 2t^3 + t)m_0 + (t^3 - t^2)m_1, \tag{7.7}$$

where $t \in [0,1]$. Figure 7.1 shows an example curve where the support points have been highlighted.

7.3.2 Extrusion

So far we have only discussed how to determine the support points of a line and the points in between. However, this is not enough for the representation

[2] Cubic Bezier and Cubic Hermite interpolation are mathematically equivalent and can be translated into each other.

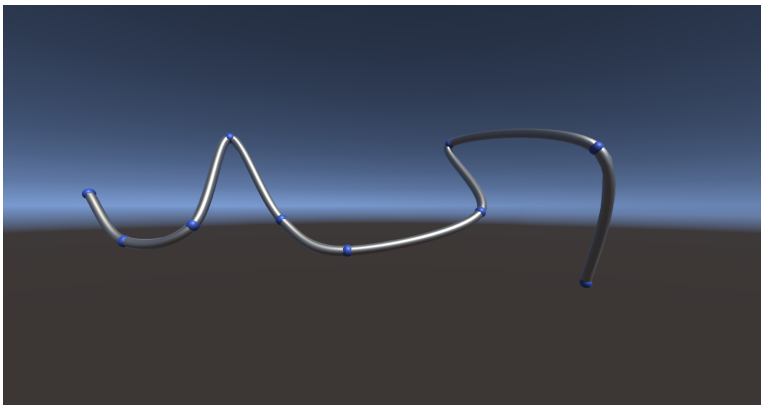


Figure 7.1: Piecewise Hermite curve with 10 control points and 9 segments

in a 3D environment. Here, lines must be created that have a volume, even if it is very small. Typically, tube-shaped geometries are created during 3D sketching, but there are also other possibilities, which will be discussed in the following.

The basic idea of extrusion is to create a three-dimensional body in which a two-dimensional basic shape is drawn along a path through space. An analogy in the real world is for example the creation of soap bubbles. Here, the tube soaked in lye is the basic form from which the (albeit slightly dented) soap bubble is extruded as soon as the tube is moved or air blows through it. Theoretically, basic shapes can be designed arbitrarily, but should not intersect itself. They can either be defined in advance and read in via configuration files or drawn interactively by the user, so that they can configure their own drawing tools. Extrusion is a basic modelling technique of CAD. While extrusion paths are precisely parametrically described in CAD, 3D sketching uses freehand pencil movements for extrusion. The basic shape required for extrusion is typically described by a list of two-dimensional points as shown in Figure 7.2(a). These are projected into three-dimensional space at each support point of the line in such a way that the respective support point and the centre of the basic shape are superimposed. In a further step, the points created are linked with the points of the last extrusion step to create displayable 3D elements (see tessellation below). There are several approaches to the question of how the basic shape should be oriented during extrusion.

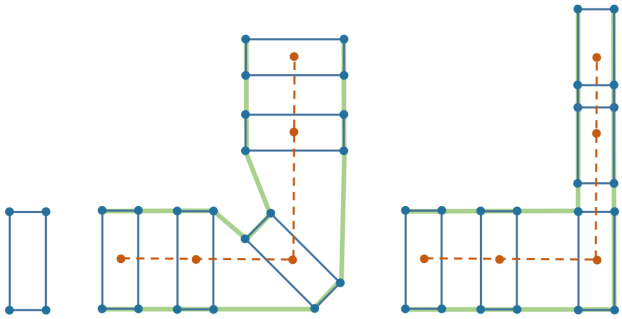


Figure 7.2: Extrusion: (a) basic shape, (b) extrusion perpendicular to the tangent of the sketched line, (c) calligraphic extrusion

7.3.2.1 Extrusion while maintaining the diameter of the basic shape

If the diameter of the basic shape is to be retained as far as possible, it must be extruded perpendicular to the tangent of the sketched line. In the case of very sharp lines, the extruded geometry may intersect itself and therefore a uniform diameter cannot be guaranteed, however, this method usually gives satisfactory results as shown in Figure 7.2(b).

7.3.2.2 Calligraphic extrusion

For other applications it may be necessary to transfer the 3D orientation of the stylus one-to-one to the extruded geometry. Similar to a calligraphic pen, different geometric shapes can be extruded depending on the direction of movement. This is typically achieved by transforming the two-dimensional basic shape in each extrusion step into the respective local coordinate system of the interaction device as shown in Figure 7.2(c).

7.4 Processing of further user input

As mentioned in Section 8.2.2.1, besides the position and orientation of the drawing tools, other input parameters can be used to influence the shape of a sketch. For example, it is possible to use a force sensor to measure how hard the user presses the pen and to determine the width of the drawn line by scaling the basic shape on its y-axis [cf. 21]. There are no limits to creativity

when mapping the input parameters to the created shape. It is possible, for example, to arrange pressure sensors radially on the pen and to manipulate the extruded basic shape according to the pressure measurements so that the user has even more control over the created shape while drawing. The speed of the pen movements can be used as a further input parameter for geometry creation. For example, fast movements can create narrow and slow movements thick shapes or, as with brush techniques, denser or looser shapes.

7.5 Tessellation and rendering

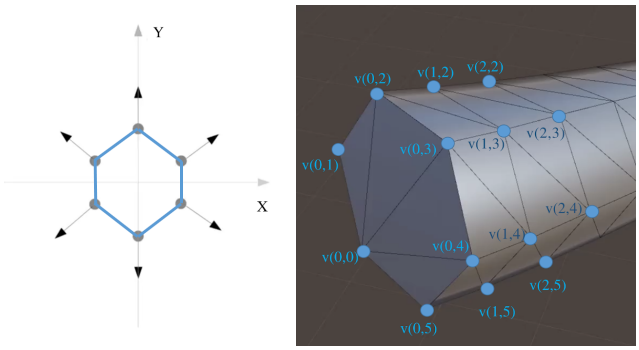
Parametric curves can have infinite resolution. In the tessellation process, polygons (usually triangles) are created from this continuous description in order to output them to the display via the 3D graphics pipeline. The smaller the step size in the tessellation process, the finer the resolution of the 3D geometry, but also the more triangles are created, which can result in performance losses.

The tessellation process can be performed on the GPU (available through DirectX 11 and OpenGL 4.0) or on the central processing unit (CPU). Although the first method is faster, the second method is often chosen for compatibility reasons.

When generating line geometries, the tessellation and extrusion processes are often closely related. First, in each extrusion step, a point of the basic shape is transformed into 3D space as described above. Then the points of the basic shape (vertices) of the current extrusion step and the previous extrusion step have to be linked in such a way that representable geometries are created. For this purpose, a mesh is created that consists of 3D points (vertices), connections (edges) and surfaces (usually triangles). Triangles are defined by specifying the indices of the points to be connected.

If a sketch basic shape with n vertices is given and $v(i, 0 \ldots n-1)$ describe the vertex indices of the current basic shape and $v(i-1, 0 \ldots n-1)$ the vertex indices of the basic shape of the previous extrusion step, the required triangles can be created in a loop:

```
// create triangles between all vertex-pairs except
// between the first and the last
for (a = 0; a < n-2; a++) {
    create_triangle( v(i, a), v(i-1, a), v(i, a+1));
    create_triangle( v(i, a+1), v(i-1, a), v(i-1, a+1));
}
```



(a) (b)

Figure 7.3: 3D extrusion: (a) basic shape consisting of six points and (b) extrusion of the basic shape.

```
// close the band: create triangles between the first
// and the last vertex pairs
create_triangle( v(i, n-1), v(i-1, n-1), v(i, 0));
create_triangle( v(i, 0), v(i-1, n-1), v(i-1, 0));
```

The result could look like Figure 7.3. If one sets $n = 2$, the `for` loop is ignored and a flat ribbon is generated. This is useful if one wants to draw lines without thickness. Note that the controller orientation can still be used to align the ribbon.

The points of the triangles can be defined clockwise or counterclockwise. This defines which of the sides of the triangle point outwards and which point inwards. In OpenGL, by default, the counterclockwise sides of the triangle point outward.

For the first (when the user presses the draw button) and last extrusion step (when the user releases the button), a cover should also be drawn on the beginning and end of the stroke so that it appears as a closed body as shown by the four triangles between the points $v(0,0) \ldots v(0,5)$ in Figure 7.3.

It is also important to set a surface normal for each vertex. Otherwise the sketched lines will be displayed in one colour and lose all shading.

The procedure described here is particularly easy and quick to implement, but can produce unattractive results in the case of unfavourable curvatures and self-penetration. More advanced subdivision techniques that adapt the shape of the triangles to the actual shape of the curve at higher resolutions are, for

example, Adaptive Tesselation and Fast Catmull-Clark Tesselation. These are explained in detail for example in Akenine-Möller et al. [19].

7.6 Modelling kernels

As an alternative to the methods presented here, some systems [e.g. 22, 23] outsource the internal geometry presentation as well as the tessellation process to external modelling kernels. Open source modelling kernels such as OpenCascade [16] are often used for this purpose. These offer the possibility to parametrically describe complex geometric bodies and to tessellate them with arbitrary levels of detail. This can be used, for example, to provide objects with different levels of detail depending on the distance to the viewer. In addition, they also support the import of CAD files, which allows them to be displayed within the 3D scene. This point is particularly advantageous when the sketching functionality is to be integrated into a CAD infrastructure.

A disadvantage of external modelling kernels is that they often require high integration and maintenance efforts. The decision to develop one's own lightweight modelling kernel or to integrate an external modelling kernel should, therefore, be made taking into account the available development capacities, the requirements for interoperability and the expected complexity of the sketches produced.

7.7 Summary

In this chapter, methods for processing freehand user interactions and techniques for transforming them into 3D sketches were described. However, only basic principles could be explained. Further issues, such as texturing or editing of 3D sketches were not mentioned. For more detailed descriptions, interested readers are referred to related literature from the field of computer graphics [e.g. 17, 18, 19].

The principles presented in this chapter can help to better understand the functionalities of the 3D sketching applications presented later in Chapter 9. Prior to this, Chapter 8 explains which interaction techniques and devices are available to users to create 3D sketches.

References

[1] R. Dachselt and A. Hübner. A Survey And Taxonomy Of 3d Menu Techniques. In *Proceedings of the 12th Eurographics conference on Virtual Environments*,

EGVE'06, page 89–99, Lisbon, Portugal, 5 2006. Eurographics Association. ISBN 978-3-905673-33-3. [Online; accessed 2020-03-07].

[2] J. J. LaViola, E. Kruijff, R. P. McMahan, D. Bowman, and I. P. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley, Boston, 2 edition, 4 2017. [Online; accessed 2020-03-07].

[3] B. Jackson and Daniel F. Keefe. From Painting to Widgets, 6-DOF and Bimanual Input Beyond Pointing. In *VR Developer Gems*. A K Peters/CRC Press, Boca Raton, FL, USA, 2019. doi: 10.1201/b21598-14.

[4] R. Hartley, J. Trumpf, Y. Dai, and H. Li. Rotation Averaging. *International Journal on Comput Vision*, 103:267–305, January 2013. doi: 10.1007/s11263-012-0601-0.

[5] Y. Lee, I. Jang, and D. Lee. Enlarging Just Noticeable Differences Of Visual-proprioceptive Conflict In Vr Using Haptic Feedback. In *2015 IEEE World Haptics Conference (WHC)*, pages 19–24. 2015 IEEE World Haptics Conference (WHC), 6 2015. doi: 10.1109/WHC.2015.7177685. ISSN: null.

[6] NIST/SEMATEC. Double exponential smoothing, 2012. URL https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc433.htm. [Online; accessed 2020-03-09].

[7] D. Keefe, R. Zeleznik, and D. Laidlaw. Drawing on Air: Input Techniques for Controlled 3D line Illustration. *IEEE Transactions on Visualization and Computer Graphics*, 13(5): 1067–1081, 2007.

[8] R. Arora, R. H. Kazi, F. Anderson, T. Grossman, K. Singh, and G. Fitzmaurice. Experimental Evaluation of Sketching on Surfaces in VR. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 5643–5654, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346559. doi: 10.1145/3025453.3025474. URL https://doi.org/10.1145/3025453.3025474.

[9] M. D. Barrera Machuca and W. Stuerzlinger. The Effect of Stereo Display Deficiencies on Virtual Hand Pointing. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300437. URL https://doi.org/10.1145/3290605.3300437.

[10] A. U. Batmaz, M. D. Barrera Machuca, D. M. Pham, and W. Stuerzlinger. Do Head-Mounted Display Stereo Deficiencies Affect 3D Pointing Tasks in AR and VR? In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 585–592. IEEE, 2019.

[11] Y. Thiel, K. Singh, and R. Balakrishnan. Elasticurves: Exploiting Stroke Dynamics and Inertia for the Real-time Neatening of Sketched 2D Curves. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, page 383–392.

ACM, 2011. doi: 10.1145/2047196.2047246. URL http://dl.acm.org/citation.cfm?id=2047246. [Online; accessed 2016-09-20].

[12] D. H. Douglas and T. K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. In *Classics in Cartography: Reflections on Influential Articles from Cartographica*, page 15–28. Wiley, Hoboken, New Jersey, USA, 2011. ISBN 978-0-470-68174-9. DOI: 10.1002/9780470669488.ch2 PMID: 11751234.

[13] C.E. Shannon. Communication In The Presence Of Noise. *Proceedings of the IEEE*, 86(2):447–457, February 1998. ISSN 1558-2256. doi: 10.1109/JPROC.1998.659497. Conference Name: Proceedings of the IEEE.

[14] Unity. Scripting API: LineRenderer, 2020. URL https://docs.unity3d.com/ScriptReference/LineRenderer.html.

[15] F. Fu. Design and Analysis of Complex Structures. In Feng Fu, editor, *Design and Analysis of Tall and Complex Structures*, pages 177–211. Butterworth-Heinemann, 1 2018. ISBN 978-0-08-101018-1. URL http://www.sciencedirect.com/science/article/pii/B978008101018100006X. DOI: 10.1016/B978-0-08-101018-1.00006-X.

[16] OpenCascade. Open CASCADE technology, the open source 3D modeling libraries | collaborative development portal, 2018. URL https://dev.opencascade.org/. [Online; accessed 2018-01-24].

[17] J. D. Foley, A. Dam, and S. K. Feiner. *Computer Graphics: Principles and Practice*. Addison-Wesley, Upper Saddle River, New Jersey, 3 edition, July 2013. ISBN 978-0-321-39952-6.

[18] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Levy. *Polygon Mesh Processing*. A K Peters, Natick, Mass, October 2010. ISBN 978-1-56881-426-1.

[19] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire. *Real-time Rendering*. Taylor & Francis, CRC Press, Boca Raton, fourth edition edition, 2018. ISBN 978-1-138-62700-0.

[20] Eric W. Weisstein. Bézier curve, 2020. URL https://mathworld.wolfram.com/BezierCurve.html. source: mathworld.wolfram.com publisher: Wolfram Research, Inc.

[21] J. H. Israel, E. Wiese, M. Mateescu, C. Zöllner, and R. Stark. Investigating Three-dimensional Sketching For Early Conceptual Design—results From Expert Discussions And User Studies. *Computers & Graphics*, 33(4):462 – 473, 2009. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2009.05.005. URL http://www.sciencedirect.com/science/article/pii/S0097849309000855.

[22] P. Fehling, F. Hermuth, J. H. Israel, and T. Jung. Towards Collaborative Sketching in Distributed Virtual Environments. In *Kultur und Informatik*, page 253–264, Glückstadt, 2018. vwh Verlag Werner Hülsbusch.

[23] VENTUS. Virtual environment for teamwork and ad-hoc collaboration between companies and heterogeneous user groups, 2017. URL http://ventus3d.com/. [Online; accessed 2017-11-09].