A Non-Photorealistic Rendering Technique for Art-directed Hatching of 3D Point Clouds

Ronja Wagner, Ole Wegen[a], Daniel Limberger[b], Jürgen Döllner and Matthias Trapp[c]

Hasso Plattner Institute, Faculty of Digital Engineering, University of Potsdam, Germany

Keywords: 3D Point Clouds, Non-photorealistic Rendering, Real-time Rendering, Art-directed.

Abstract: Point clouds or point-based geometry of varying density can nowadays be easily acquired using LiDAR cameras or modern smartphones with LiDAR sensors. We demonstrate how this data can be used directly to create novel artistic digital content using Non-Photorealistic Rendering techniques. We introduce a GPU-based technique for art-directable NPR rendering of 3D point clouds at interactive frame-rates. The technique uses either a subset or all of the points to generate oriented, sketchy strokes by taking local curvature and normal information into account. It uses X-Toon textures as part of its parameterization, supports hatching and cross hatching, and is inherently temporal coherent with respect to virtual camera movements. This introduces significant artistic freedom that is underlined by our results, which show that a variety of different sketchy styles such as colored crayons, pencil, pointillism, wax crayons, blue print, and chalk-drawings can be achieved on a wide spectrum of point clouds, i.e., covering 3D polygonal meshes as well as iPad-based LiDAR scans.

1 INTRODUCTION

Point clouds are a simple, compact, and flexible geometric representation that can easily be acquired for real-world scenes using off-the-shelf photogrammetric techniques. With the introduction of Light Detection And Ranging (LiDAR) sensors in high-end smartphones, e.g., iPhone 12, straightforward acquisition of point clouds is easily accessible to consumers.

Despite both its potential applicability in various domains and increasing popularity in social media—mainly for 3D photography and animation with the point cloud as an art style in itself— their usage as a fundamental geometric representation in Non-Photorealistic Rendering (NPR) as well as respective rendering systems or frameworks are sparsely studied. Notable uses of 3D point clouds are (1) to enable light transport for rendering scanned assets (Sabbadin et al., 2019), (2) stylized, interactive self portraits, (3) data visualization, as well as (4) art-directed particle animation and stylization such as the reprojection of recorded memories in the computer game Cyberpunk 2077 (Ankermann et al., 2021) as a means of distinguishing between reality and virtual reality. Another noteworthy use can be found in previsualization or
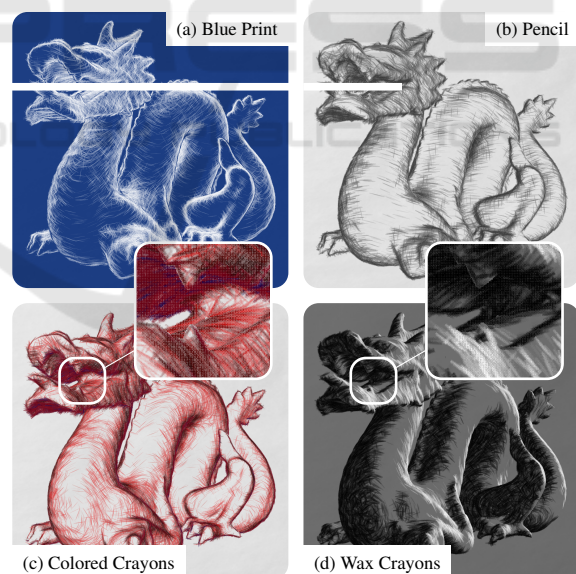

Figure 1: The *Stanford dragon* as 3D point cloud, rendered in four different styles using art-direct hatching.

digital content creation to evaluate ideas and concepts early before performing possibly costly data enhancement or transformation, e.g., creating stylized, textured 3D meshes. We present a first approach into this domain by researching methods to enable art-directed procedural-controlled rendering effects.

[a] https://orcid.org/0000-0002-6571-5897
[b] https://orcid.org/0000-0002-9111-4809
[c] https://orcid.org/0000-0003-3861-5759

In the context of NPR, point clouds have both advantages and disadvantages over traditional 3D polygonal models. Point clouds lack inherent connectivity information that needs to be respected or maintained, which simplifies their representation, transformation, and rendering. The absence of surface-related attributes such as normal vectors, texture coordinates, or curvature data is rather inconvenient, since they are essential for the implementation of NPR techniques. Additionally, while high quality renderings of point clouds are possible (Schütz and Wimmer, 2015; Schütz et al., 2021), they often rely on a sufficient point density. With these observations in mind, enabling the implementation of real-time, art-directed NPR techniques comprises the following major challenges:

Art-directable Parametrization (C_1): Decoupling of processing operations that are required for different manifestations of stylization techniques, e.g., point-splat functions (Anjos et al., 2018), to facilitate the interchange of respective techniques and increasing ease-of-use.

Efficient Data Handling (C_2): An efficient representation of complex point clouds and respective stylization data to allow for art-directed stylization based on point-attribute data. This also concerns the reduction of data transfer and update latency to support real-time rendering for interactive control.

Existing approaches to implement animation techniques for point clouds are (1) customized integration into existing game engines (to enable real-time rendering) or (2) specific tooling or scripts to extend existing software, e.g., Blender. These, however, either require auxiliary constructs for data representation, e.g., texture-based representations, lack real-time rendering capabilities, or can hardly handle millions of points.

In the context of the challenges C_1 and C_2, we present a Graphics Processing Unit (GPU)-based hatching technique for producing art-directed non-photorealistic renderings of potentially massive point clouds, which are obtained either using LiDAR scanning of a real-world scene or by sampling polygonal geometry. The presented approach is not limited to hatching, but can serve as a general approach for the implementation of further non-photorealistic rendering techniques for point clouds.

To this end, we...

1. ...introduce a method for creating sketched images directly out of 3D point clouds using only position, normal, and curvature information,

2. enable significant artistic freedom due to wide range of parameters by using X-Toon textures, and

3. demonstrate the method's real-time capability (precomputed attributes) and temporal coherence.

2 RELATED WORK

Prior art with respect to texture-based stylization in 3D object space using, e.g., proxy-geometry as well as in image space remains quite popular and is well covered. In the following, we briefly classify an exemplary cross section of classics as well as latest works.

NPR rendering of 3D polygonal meshes targeting a sketched look and feel can be easily achieved using texture-based hatching approaches (Praun et al., 2001; Webb et al., 2002). Hatching patterns are pre-generated in multiple levels of details, i.e., *Tonal Art Maps*, and enable fine control over shading without aliasing. In order to also create sketchy outlines or silhouettes, image-based approaches can be applied (Nienhaus and Döllner, 2003; Zakaria and Seidel, 2004). These types of techniques, however, cannot be applied to point clouds, since texturing and image-based post processing is implemented fundamentally different; Strokes are placed directly and individually, and require a dedicated level of detail concept. In addition, image-based curvature estimates (Kim et al., 2008) and purely image-based stippling (Awano et al., 2010) are not applicable and, regardless, usually lead to temporal incoherent results.

Another common approach for simulating, e.g., pencil-like renderings, is to use edges of the 3D meshes to create overlapping graphite strokes (Sousa and Buchanan, 1999). This concept of creating textured proxy geometry, also sometimes referred to as graphtals (Smith, 1984), facilitates temporal frame-to-frame coherence and can be applied to particles and point clouds alike (Kaplan et al., 2000; Markosian et al., 2000). The proxy geometry, e.g., dots or strokes, can be specialized dynamically at run-time w.r.t. size, orientation, placement, and color. Thereby, artistic control can be linked to mesh properties and thus allow for silhouettes and contouring (Grabli et al., 2004). This approach represents the core of our technique and, apart from various differences in the implementation details due to the evolution of graphic hardware, can be easily extend, enhanced, and specialized for point clouds.

A promising extension to this are *ribbons*, i.e., one-dimensional strips of points, ordered "according to their neighbourhood relations" (Runions et al., 2007). The concept could be used to create silhouette or curvature aware proxy geometry (Schmid et al., 2011) such as strokes consisting of multiple links instead of single, straight ones, and might allow for fine-grained stippling styles. So far, existing approaches covering water color, caligraphy, etc. still rely on strokes drawn by the artist (Zheng et al., 2017). Automatic splatting of lines or strokes, though applicable to volumetric

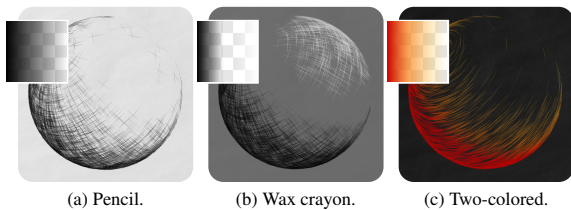(a) Pencil. (b) Wax crayon. (c) Two-colored.

Figure 2: Shading effects and their corresponding X-Toon textures (RGBA). Note that the y-axis is not utilized here.

data as well, does only cover a specific style and is not easily art-directable (Zhang et al., 2014; Anjos et al., 2018). We partially combine these ideas and aim for an automated approach that allows for a constrained but artist-directable, expressive stylization.

3 ART-DIRECTED HATCHING

This section details concepts as well as implementation details of our real-time hatching technique.

3.1 Art-directed Parametrization

We make the assumption that any point cloud is given with point normal vectors (Cao et al., 2021), color, and curvature attributes (Mérigot et al., 2009; Lu et al., 2020). The latter two can improve the appearance but are not required. To achieve the desired hatching effect, each point is replaced by textured proxy geometry, e.g., two orthogonal lines aligned to the surface, textured with pencil strokes.

A drawing technique that characterizes cross-hatching is to depict lighting by not only using different colors but also by omitting lines and therefore showing the underlying paper. The amount of light that reaches a point defines how likely it is to be omitted. To accomplish a wide variety of artistic styles and effects, the color and point density for each amount of light can be controlled through the alpha channel of an X-Toon (Barla et al., 2006; Kang et al., 2009) texture (Fig. 2). On white paper for example, strongly illuminated parts are realized by removing most points, while darker parts keep their original point density. Another option (better suited to darker backgrounds) is to color well-lit points brightly and leave out points receiving a medium amount of light.

Lighting by omission only works if there are no visible parts behind the points that are left out. To avoid such parts, e.g., the back of the model, from appearing where only the paper should be visible, we use an additional depth pass. With that pass we ensure that nothing behind the surface of the model is visible in the final result.

For a more realistic pencil effect there is also the option to limit the stroke colors to a number of pre-defined pencil colors, similar to a colored pencil set. In addition to the classic X-Toon parametrization, the following parameters are provided for hatching:

Line Length and Thickness: These parameters control the extent of hatching strokes.

Parallel to cross-hatch Proportion: Our technique supports both cross hatching, where almost all points are replaced with two lines, and parallel hatching, where all points are replaced with a single line. This parameter specifies the probability for a point to be replaced with a single stroke or two, crossed strokes.

Stroke Density: In point clouds of high density, the individual hatched strokes might not be distinguishable from each other. By specifying the proportion of strokes to be left out, the density of rendered strokes can be controlled.

Line Direction: This parameter is used to determine the direction of each stroke.

Light Direction: For lighting we use a uniform light direction that is specified by this parameter.

Coloring Mode: Defines which method is to be used for determining the color of the hatched strokes.

Hue Shift: Is applied to the color calculated with the selected coloring mode.

3.2 Stroke Generation for Hatching

X-Toon Coordinate Computation. To determine the desired density and therefore the proportion of points to be omitted, we first compute the X-Toon texture coordinates, namely depth and brightness, for a point. Transforming the point's z-coordinate (after projection) into a $[0,1]$ range yields the depth coordinate. The brightness coordinate depends on the scene lighting implementation, e.g., Phong-based lighting (Phong, 1975). With these coordinates and the corresponding texture value, we determine whether the current point should be discarded or replaced by lines, and in the latter case, obtain the color of these lines.

Probabilistic Interpolation. Both lighting by omission and the pencil color quantization rely on a probabilistic approach. Instead of interpolating between two values (e.g., visible and not visible or two RGB values) and applying the result, the point only attains one of the values chosen at random with the interpolation factor determining the probabilities. With a sufficient number of points, the result creates an approximate
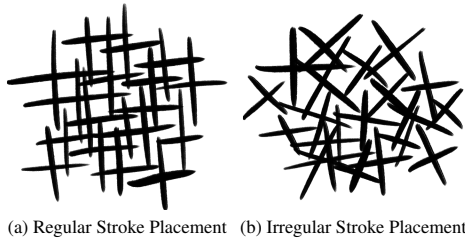
(a) Regular Stroke Placement   (b) Irregular Stroke Placement

Figure 3: For hatching, one or more strokes, i.e., two for cross-hatching, can be placed and oriented either in a (a) regular more accurate or (b) irregular and scattered way.

impression of directly interpolated values. As random seed, we use the point ID, which is temporally consistent. For determining whether a point should be visible or not, we use the alpha value from the X-Toon texture as the interpolation factor between visibility and invisibility. So if $a \in [0;1]$ is the alpha value, the point has a $100 \cdot a\%$ chance of being visible. The general point density can be controlled through a corresponding parameter that we multiply with the alpha value from the X-Toon texture.

Line Geometry Generation. For the cross hatching effect we replace each point with two orthogonal lines perpendicular to the point's normal. In doing so, we need to ensure the line directions are consistent for all points. Line directions are consistent if points with similar normal vectors produce similar line directions (Fig. 3) so that they properly illustrate the surface direction. To achieve that consistency we project a normalized uniform vector on the plane defined by each point normal. Let $\vec{u}$ be the uniform vector and $\vec{n}$ be the surface normal. The projected vector $\vec{d_1}$ is computed as follows: $\vec{d_1} = \vec{u} - (\vec{u} \cdot \vec{n}) \cdot \vec{n}$. The second direction $\vec{d_2}$ can be determined by simply computing the normalized cross product of the first direction and the point normal: $\vec{d_2} = \vec{n} \times \vec{d_1}$. With these directions, we can now create two orthogonal lines along the surface. Let p be the point coordinates and l be the line length parameter. Then the first line extends from $p - \vec{d_1}$ to $p + \vec{d_1}$. The second line starts at $p - \vec{d_2}$ and ends at $p + \vec{d_2}$. We can now use these coordinates to create rectangle geometry for each line. One side of said rectangle is described by the corresponding line direction and length. The direction of the other side needs to be orthogonal to both the line direction and the vector from the camera to the current point and can therefore be calculated by taking their cross product. The width of the line is defined in a separate line thickness parameter. From this information, we can derive the four vertices of the resulting rectangle, with the texture coordinates $(0,0)$, $(0,1)$, $(1,1)$ and $(1,0)$ for the pencil stroke texture.

3.3 Local Shading Models

There are three ways to determine the color of the created pencil strokes. The simplest one is to apply the same color to all strokes, which can, for example, be used to create a classic gray pencil sketch or a blue ballpoint pen drawing. Alternatively, the color can be taken directly from the X-Toon texture. The third approach is to blend the initial point color with the X-Toon texture value to depict both lighting and the original color information.

After the point color is determined, it is possible to add pencil color quantization. That means only allowing colors from a predefined set to simulate drawing with pencils from a limited pencil box. Unlike other media, pencil colors can only be mixed to a limited extent. With watercolors for example, three colors and black and white are in theory sufficient to get every other color. For colored pencils however, you need more base colors to illustrate the whole color spectrum properly. This is because generally, the quality of the blending result of two pencil colors suffers with decreasing similarity of the blended colors.

Our pencil color effect simulates that by calculating the two closest colors from a predetermined pencil set to a given color and blending them with probabilistic interpolation. We call the given color c_0. The pencil color set is given as rgb and hsv values sorted by hue. It contains only colors with high saturation and value. First, we calculate the hsv representation of c_0 and apply the hue shift parameter. We can then search for the colors c_1 and c_2 from the pencil set whose hues are closest to c_0's hue, so that $\text{hue}(c_0) = \text{hue}(c_1) \cdot a + \text{hue}(c_2) \cdot (1-a)$ applies for an $a \in [0,1]$. We can use this a as an interpolation factor for probabilistic interpolation to create the impression of a color with the same hue as c_0. Finally, we mix the resulting color with black and white to more closely match c_0 in value and saturation.
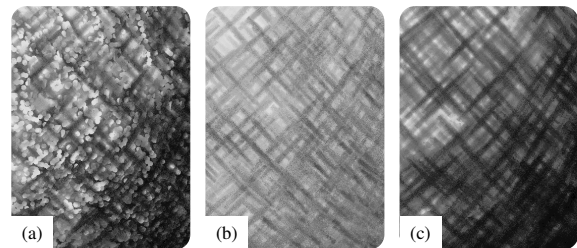

(a)   (b)   (c)

Figure 4: The point sprites used during the depth pass cause discontinuities in the z-buffer which interfere with the strokes rendered later (a). In addition, the semi-transparent strokes interfere with one another (b). By shifting the strokes towards the virtual camera using polygon offsets while using order-independent rendering (blending or alpha to coverage) both effects can be circumvented respectively (c).

3.4 Depth Pass

For the lighting by omission approach to work, we need to ensure there are no visible lines behind the points that are omitted. A first approach to that problem is back face culling, which prevents the back of the surface from being visible. However, even with back face culling, there still are lines behind the surface that are not supposed to be visible (Fig. 6). We prevent that by using an additional rendering pass before the hatching pass.

In the depth pass, we render every point as a single circle and write only to the depth buffer. With a sufficient point density and point size, these circles cover the whole surface, so in the following hatching pass, only lines in front of the surface get rendered. Without any modifications however, the circles of the points at the front will cover the pencil strokes of nearby points. To avoid that problem, we add an offset to the polygons in the hatching pass, so the pencil strokes are slightly in front of their corresponding points from the depth pass (Fig. 4 a). Since the depth test needs to be activated in the drawing pass for this approach to work, we need to make sure nothing is written to the depth buffer in the hatching pass, otherwise the lines will cover each other instead of overlapping and thereby creating darker tones (Fig. 4 b).

The size of the circles is also controllable via a parameter and needs to be adapted to achieve the desired result in the depth pass and to not cover too much or to little of the hatched lines.

3.5 Enhancements

The approach described so far allows for decent results, though the following tweaks should be considered.

Alpha Modification. So far, we use the approach of probabilistic interpolation to create the impression of the surface being transparent. However, for image regions with almost full transparency, this results in very few strong strokes on an otherwise clear part of the surface, when in an actual pencil sketch, you would also adjust the pressure of the pencil to modify the transparency of the strokes themselves. That is

why we adjust the alpha value of the generated pencil stroke proportional to the alpha value from the X-Toon texture. Also, even in parts with similar lighting, not every stroke is applied with exactly the same pressure. To account for that, we apply some randomness on the final alpha value.

Interpolation of Cross & Parallel Hatches. In addition to a decreased stroke strength, another technique to convey higher transparency is to use fewer lines by transitioning from cross hatching to parallel hatching. We implement this approach by using the alpha value from the X-Toon texture as the probability for replacing a point with a cross instead of a line. We multiply that probability by a parameter $h \in [0, 1]$, allowing us to influence the proportion of cross-hatched to parallel-hatched points. This parameter also makes it possible to create results that exclusively use parallel hatching.

Curvature-controlled Line-length. So far, we replace each point with equally long, straight lines. Put together, they already create a good illustration of curved surfaces because the lines are short enough that they are a good approximation of the direction of the surface below them. On areas of high curvature on the other hand, the surface direction changes faster. That means a line that is as long as the lines on low-curvature areas is not a good surface approximation anymore. With curvature information at each point, we can align for that by decreasing the length of lines in high-curvature areas (Fig. 5). In addition to line length modification through curvature, we also modify the line length with a randomness factor to account for the differences in line length when drawing by hand.

4 RESULTS & DISCUSSION

This section present and discusses various application examples achieved by using our art-directable rendering technique.

Application Examples. Fig. 7 shows an overview of different point cloud stylization variants obtained by our technique. The variety covers a range from blueprint rendering, gray-scale charcoal and ballpoint pens to classic pencil drawing or pointillism. Our technique allows for modification of all provided stylization parameters and presents them interactively to support the demands of artists.

Performance Considerations. We tested the rendering performance of a prototypical implementation
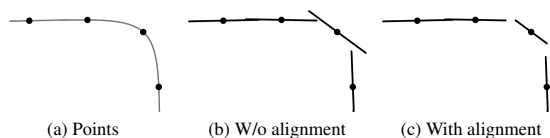

Figure 5: The length of strokes can be aligned based on the local curvature at the stroke's location. This can be used to achieve (b) increased sketchiness especially in curved areas as well as (c) a strict, more accurate hatching.

(a) Naive Rendering   (b) Back Face enabled   (c) Depth Pass enabled   (d) Difference of (b) and (c)
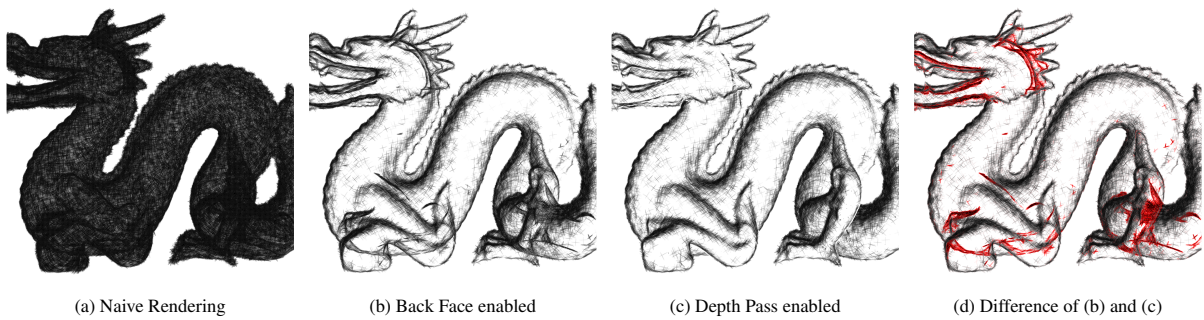
Figure 6: The *Stanford dragon* rendered as 3D point cloud (a) without back face culling, (b) with back face culling, and (c) with an additional depth pass. Points—or strokes—that are hidden due to the depth pass are highlighted in red in (d).

using three point clouds of different complexity ranging from approx. 3.5 million points (LiDAR, Statue) over 5.7 million points to 11 million points (Angel statue, sampled 3D mesh). The first point cloud with 3.5 million points resulted in a frametime of about 30 milliseconds, with 5.7 million points we measured 60 milliseconds and for 11 million points we obtain a frametime of 100 milliseconds. The performance test was conducted using an AMD Radeon 5700XT with 8 GB VRAM on a Ryzen 5 CPU with 3.6 GHz and 32 GB RAM rendering at a viewport resolution of 2947×1661 pixels. The run-time performance depends mainly on the geometric complexity of the 3D scene and decreases approx. linearly with the number of points to process. However, while a high point density is useful for the depth pass to avoid gaps between points it is not conducive to the overall visual quality of the hatching effect, i.e., the generated strokes are hardly distinguishable.

Discussion. Due to the nature of point clouds, especially when created using consumer LiDAR scanners, outliers from the depth pass (remember, all points are used for this pass) sometimes cover strokes and might not be desired. Similarly, the point cloud density is typically rather irregular, which sometimes leads to strokes bleeding from the background into the foreground. Both effects can be mitigated by tweaking the polygon offset as well as the adjusting the point size within the depth pass. Furthermore, these effects are not inherently wrong and are often also desirable for less dense point clouds.

The same applies for strokes which leave an object's expected silhouette or contour (over-shooting of strokes). For us, this was mostly visually pleasing and also a welcome differentiator from image-based renderings. There might be cases where this is not desired or not fitting a specific style though. Since point clouds are never expected to be two manifold (i.e., watertight or closed) and more often used for capturing open environments than for solid and closed objects, we render the mentioned drawbacks as non-problematic exceptions of our technique. The distinctive styles can be easily achieved with only minor configurations within our art-directed parameterization and the technique can be implemented without exotic extension or groundbreaking graphics APIs.

5 CONCLUSIONS

This paper presents a real-time non-photorealistic rendering technique for art-directed hatches of 3D point clouds. By extending the X-Toon approach of Barla *et al.*, it facilitates various configurations yielding different stylization results for potentially massive 3D point clouds that are obtained by LiDAR acquisition or sampling polygonal 3D meshes. The current state provides basis to develop and apply more advanced stylization techniques for 3D point clouds. For example, to synthesize more sophisticated stylized renderings, our technique can be extended to support outlines (Rosenthal and Linsen, 2008) or even suggestive contours (Proença et al., 2008) using the derived curvature information (Mérigot et al., 2009; Lu et al., 2020). Further, level-of-detail approaches for geometry generation stage can be implemented enabling seamless level-of-abstraction transitions (Semmo and Döllner, 2014). Furthermore, the depth pass could be improved by using oriented point splats (Anjos et al., 2018).

ACKNOWLEDGMENTS

(a) Cross Hatched

(b) Colorful Contour

(c) Classic Pointillism

(d) Two-Colored Dark

(e) White and Black Chalk

(f) Blueprint

(g) iPad Scan, Hatched

(h) iPad Scan, Cross Hatched

(i) iPad Scan, Crayons

(j) Statue

(k) Temple
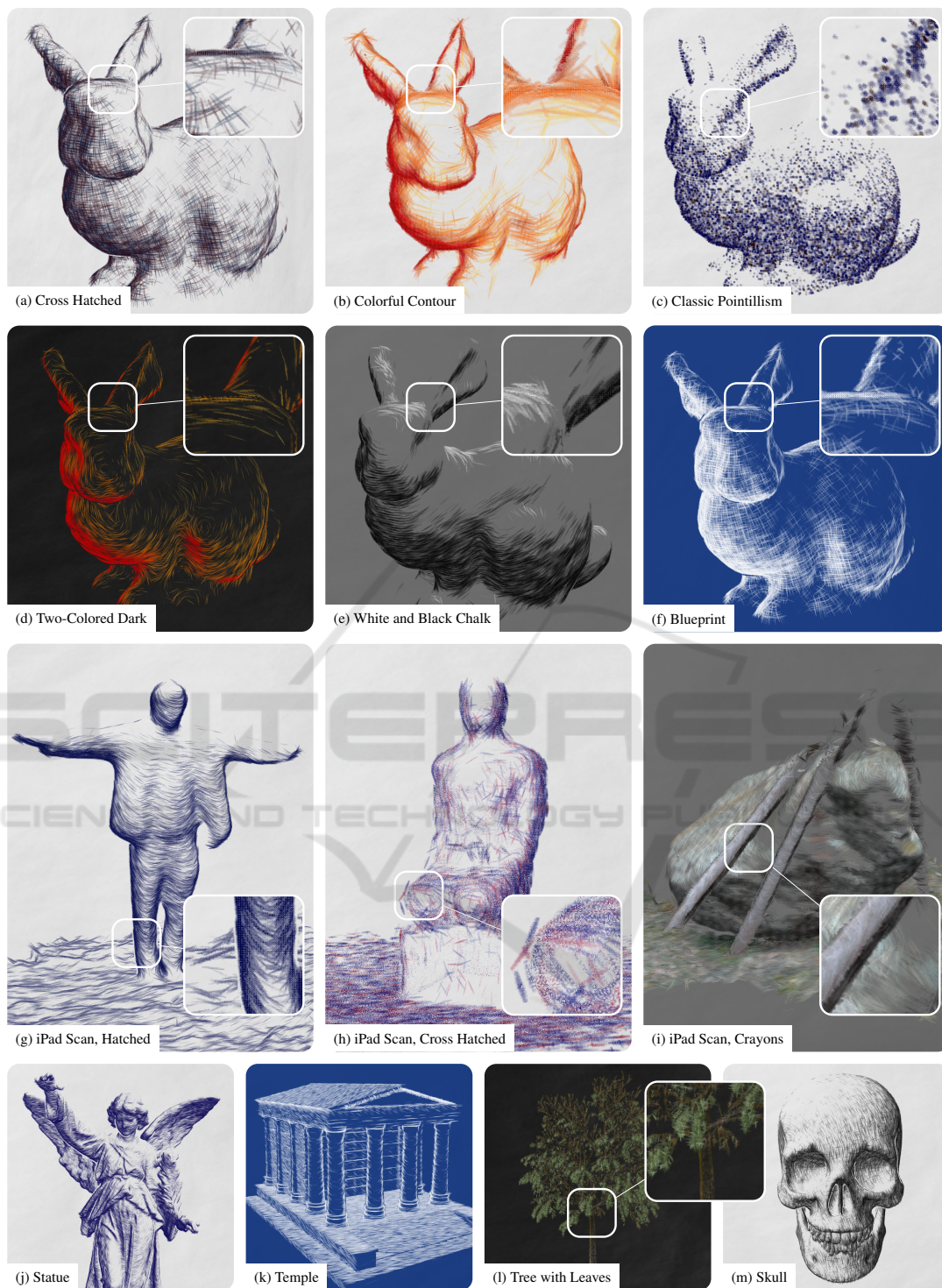
(l) Tree with Leaves

(m) Skull

Figure 7: Various results of our NPR rendering-technique. A point cloud scan of the *Stanford bunny* was used in (a) to (f). The point clouds in (g) to (i) have are actual point clouds captured using an iPad. In (j) to (m) the points are sampled from 3D polygonal meshes and rendered using our NPR technique. All results are captured from our C++ implementation, rendered interactively and in real-time (60Hz+ in Full-HD and QHD) with a temporal coherence not possible using image-based approaches. Meshes used for (j) and (k) are licensed under *CC Attribution* and have been created by *noe-3d.at* (sketchfab.com/3d-models/grabfigur-cfec3a9d71db42978d4a377a0e8f1154) and *Robin Butler* (sketchfab.com/3d-models/roman-temple-f5efe108fb19419ab985ef69d1762839) respectively.

REFERENCES

Anjos, R. K., Ribeiro, C. S., Lopes, D. S. o., and Pereira, J. a. M. (2018). Stroke-based splatting: An efficient multi-resolution point cloud visualization technique. *Vis. Comput.*, 34(10):1383—1397.

Ankermann, P., Swierad, O., and Krzyscin, K. (2021). The tech and art of cyberspaces in cyberpunk 2077. In *ACM SIGGRAPH 2021 Talks*, SIGGRAPH '21, pages 21:1–2. Association for Computing Machinery.

Awano, N., Nishio, K., and Kobori, K.-i. (2010). Interactive stipple rendering for point clouds.

Barla, P., Thollot, J., and Markosian, L. (2006). X-toon: An extended toon shader. In *Proc. ACM NPAR*, NPAR '06, page 127–132. Association for Computing Machinery.

Cao, X., Shi, B., Okura, F., and Matsushita, Y. (2021). Normal integration via inverse plane fitting with minimum point-to-plane distance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2382–2391.

Grabli, S., Turquin, E., Durand, F., and Sillion, F. X. (2004). Programmable style for npr line drawing. In Keller, A. and Jensen, H. W., editors, *Eurographics Workshop on Rendering*. The Eurographics Association.

Kang, D., Chung, J.-M., Seo, S.-H., Choi, J.-S., and Yoon, K.-H. (2009). Detail-adaptive toon shading using saliency. In *2009 Second International Conference in Visualisation*, pages 16–20.

Kaplan, M., Gooch, B., and Cohen, E. (2000). Interactive artistic rendering. In *Proc. 1st International Symposium on Non-Photorealistic Animation and Rendering*, pages 67–74.

Kim, Y., Yu, J., Yu, X., and Lee, S. (2008). Line-art illustration of dynamic and specular surfaces. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 156:1–10. ACM.

Lu, D., Lu, X., Sun, Y., and Wang, J. (2020). Deep feature-preserving normal estimation for point cloud filtering. *CoRR*, abs/2004.11563.

Markosian, L., Meier, B. J., Kowalski, M. A., Holden, L. S., Northrup, J. D., and Hughes, J. F. (2000). Art-based rendering with continuous levels of detail. In *Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '00, pages 59—-66. Association for Computing Machinery.

Mérigot, Q., Ovsjanikov, M., and Guibas, L. (2009). Robust voronoi-based curvature and feature estimation. In *2009 SIAMACM Joint Conference on Geometric and Physical Modeling*, SPM '09, pages 1—-12. Association for Computing Machinery.

Nienhaus, M. and Döllner, J. (2003). Sketchy drawings: A hardware-accelerated approach for real-time non-photorealistic rendering. In *ACM SIGGRAPH 2003 Sketches & Applications*, SIGGRAPH '03, page 1. ACM.

Phong, B. T. (1975). Illumination for computer generated pictures. *Commun. ACM*, 18(6):311—-317.

Praun, E., Hoppe, H., Webb, M., and Finkelstein, A. (2001). Real-time hatching. In *Proc. ACM SIGGRAPH*, pages 579–584. ACM.

Proença, J., Jorge, J. A., and Sousa, M. C. (2008). Suggestive contours over point-set implicits. In *GRAPP*, pages 171–180. INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

Rosenthal, P. and Linsen, L. (2008). Image-space point cloud rendering. In *Proceedings of Computer Graphics International (CGI) 2008*, pages 136–143. Association for Computing Machinery.

Runions, A., Samavati, F., and Prusinkiewicz, P. (2007). Ribbons: A representation for point clouds. *The Visual Computer*, 23(9-11):945–954.

Sabbadin, M., Palma, G., Banterle, F., Boubekeur, T., and Cignoni, P. (2019). High dynamic range point clouds for real-time relighting. *Computer Graphics Forum*, 38(7):513–525.

Schmid, J., Senn, M. S., Gross, M., and Sumner, R. W. (2011). Overcoat: An implicit canvas for 3d painting. *ACM Trans. Graph.*, 30(4):28:1–10.

Schütz, M., Kerbl, B., and Wimmer, M. (2021). Rendering point clouds with compute shaders and vertex order optimization. *Computer Graphics Forum*, 40:115–126.

Schütz, M. and Wimmer, M. (2015). High-Quality Point Based Rendering Using Fast Single Pass Interpolation. In Guidi, G., Scopigno, R., and Brunet, P., editors, *International Congress on Digital Heritage - Theme 2 - Computer Graphics And Interaction*. IEEE.

Semmo, A. and Döllner, J. (2014). Image filtering for interactive level-of-abstraction visualization of 3d scenes. In *Proceedings International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging (CAe)*, pages 5–14.

Smith, A. R. (1984). Plants, fractals, and formal languages. In *Proc. ACM SIGGRAPH*, SIGGRAPH '84, pages 1–10. ACM.

Sousa, M. C. and Buchanan, J. W. (1999). Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics Forum*.

Webb, M., Praun, E., Finkelstein, A., and Hoppe, H. (2002). Fine tone control in hardware hatching. In *NPAR 2002: Second International Symposium on Non Photorealistic Rendering*, pages 53–58.

Zakaria, N. and Seidel, H.-P. (2004). Interactive stylized silhouette for point-sampled geometry. In *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '04, pages 242—-249, New York, NY, USA. Association for Computing Machinery.

Zhang, L., Sun, Q., and He, Y. (2014). Splatting lines: An efficient method for illustrating 3d surfaces and volumes. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '14, pages 135—-142. Association for Computing Machinery.

Zheng, M., Milliez, A., Gross, M., and Sumner, R. W. (2017). Example-based brushes for coherent stylized renderings. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '17, pages 3:1–10. ACM.