Non-Photorealistic Novel View Synthesis Using Radiance Fields

Medard Szilvasy
Supervisors: Elmar Eisemann, Petr Kellnhofer, Michael Weinmann
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Medard Szilvasy
Final project course: CSE3000 Research Project
Thesis committee: Elmar Eisemann, Petr Kellnhofer, Michael Weinmann, Jan van Gemert

Abstract

Radiance fields are a promising alternative to conventional 3D representations in the domain of novel view synthesis, with recent research achieving truly impressive photorealistic view synthesis results. In this paper, we deal with the concept of non-photorealistic rendering in the context of radiance fields, for generating more stylistic captures of real-world objects. We discuss the suitability of inverse rendering as a stepping-stone to traditional shading and edge detection, and contribute a new algorithm specifically for outline detection in radiance fields.

1. Introduction

Novel view synthesis encompasses a range of techniques that aim to allow new views of 3D scenes to be generated from a limited set of input images. The field has seen significant progress in recent years, with much of the latest research developing radiance fields as a way of representing 3D environments [3, 14, 18, 22] – modelling a function that maps each viewing direction and location in space to its volume density and colour, allowing us to reconstruct a scene's geometry and view-dependent appearance.

Radiance fields achieve state-of-the-art, photorealistic view synthesis results, but applying existing stylised rendering approaches is currently not directly possible with these representations. Traditionally, these non-photorealistic methods rely on scene properties that are also involved in photorealistic rendering [13], such as vertex positions, surface normals, texture and reflectance, but these are not an inherent part of RGB radiance fields. Additionally, many such algorithms emphasise the visible *lines* in an image, including silhouette edges, boundaries and creases [6, 9, 10, 17].

This paper explores how non-photorealistic rendering (NPR) can be adapted to radiance fields. In particular, we leverage the contributions of previous work on inverse rendering [11, 23, 25] to recover the material and geometric properties of a scene, allowing re-rendering using a variety of non-photorealistic methods. We also examine a number of edge detection algorithms and their applicability, including existing image-space algorithms, as well as introducing a new object-space algorithm that makes use of the ability of radiance fields to capture depth and shape in 3D space.

Non-photorealistic rendering is a broad field encompassing techniques ranging from artistic rendering to imitate cartoon [6], pencil sketch [13], pen-and-ink [20] and watercolour [5] styles, to cool-to-warm shading for better capturing geometric information in technical illustration [8].

Project page: `https://mszilvasy.github.io/TensoIR-NPR`

Figure 1. Some stylised novel view renderings using our approach.

While this paper demonstrates the successful application of some NPR styles, the described methodology is intended to be applicable to all 3D NPR methods that rely on the same geometric and material properties. The approach to inverse rendering does not matter either, as long as it also extracts these properties.

In summary, this paper demonstrates the application of classical non-photorealistic rendering to novel view synthesis via radiance fields, and contributes a new edge detection algorithm suitable for finding outlines in radiance fields.

2. Related Work

Recent interest in radiance fields as a means of view synthesis was spurred by the introduction of the neural radiance field (NeRF) [14], which was able to achieve high-fidelity renderings of complex scenes with view-dependent effects. NeRF trains and evaluates a multilayer perceptron (MLP) to obtain radiance field samples, and while some research since has expanded on this purely MLP-based approach [22], there are also variants which represent radiance fields as voxel grids to achieve superior training rates and reconstruction times [3, 18].

Previous work on non-photorealistic stylisation of radiance fields [15, 21, 24] focuses on applying neural style transfer [7] to combine the contents of the novel views generated by radiance fields with the artistic style of a reference image. This paper differs from such research in that it focuses on how recovering the material and geometric properties of a scene enables a range of classical (non-photorealistic) approaches to rendering and shading to be

applied to radiance fields as they are to traditional 3D representations.

Non-photorealistic rendering (NPR) covers various rendering techniques that deviate from photorealistic methods to stylise renders for artistic purposes [5, 13, 20] or visual communication [8, 9, 17]. NPR techniques use much of the same processes as the photorealistic techniques they diverge from, and the NPR methods explored in this paper borrow from, and contrast with, the photorealistic lighting models of Phong [16] and Blinn [1].

Inverse rendering deals with estimating the geometric and materials properties that contribute to the composition of an observed image. In the context of radiance fields, this has been extended to recovering the properties of a 3D environment and novel views generated from the scene [11,23,25]. The existing research demonstrates applications such as relighting and material editing, using photorealistic approaches to rendering. In this paper, we explore how these uses can be extended to non-photorealistic rendering, and the most important features for these purposes.

3. Method

In this section, we discuss a method for producing stylised novel view renderings, with non-photorealistic shading and explicitly drawn silhouette edges and creases. First, we describe various shading techniques and their applicability to radiance fields owing to the contributions of previous work on inverse rendering (Sec. 3.1). We then discuss edge detection in 2D image space to represent outlines and surface discontinuities (Sec. 3.2). Finally, we introduce our own algorithm for silhouette edge detection in 3D object space within radiance fields, a novel contribution of this paper which hopes to surpass the accuracy of aforementioned image space approaches (Sec. 3.3).

3.1. Shading

A broad range of shading algorithms, both photorealistic and otherwise, rely on the same material parameters and understanding of optics. At minimum, we require surface positions, normals, albedo (for diffuse reflection), and potentially an attribute such as shininess or roughness to regulate specular reflections. A number of so-called inverse rendering methods exist to train radiance fields for the recovery of these properties [11, 23, 25], opening the way for a variety of shading styles as we will now examine.

Blinn-Phong reflection model. We begin with a photorealistic lighting model, consisting of separate diffuse and specular components. The intensity of diffuse reflection at a surface is modulated by the cosine of the incident angle between the direction of incident light and the surface normal $\hat{\mathbf{L}} \cdot \hat{\mathbf{N}}$, as per Lambert's law. In considering the contribution of specular reflection, Phong's shading algorithm [16] uses the cosine of the angle between the direction of reflect light and the viewer's line of sight $\hat{\mathbf{R}} \cdot \hat{\mathbf{V}}$, whereas a modified approach by Blinn [1] favours the dot product between the normal and a "halfway vector", $\hat{\mathbf{N}} \cdot \hat{\mathbf{H}}$, where $\hat{\mathbf{H}} = \frac{L+V}{\|L+V\|}$. The overall Blinn-Phong reflection model for a single light source can therefore be summarised as

$$i = k_d k_a + k_d(\hat{\mathbf{L}} \cdot \hat{\mathbf{N}}) + k_s(\hat{\mathbf{N}} \cdot \hat{\mathbf{H}})^{\alpha} \tag{1}$$

where k_d and k_s represent the diffuse and specular reflectance, respectively, k_a is the ambient illumination, and α is a shininess constant for the surface.

By recovering albedo, depth, normal and shininess/roughness maps using inverse rendering (see Fig. 2b), we can directly apply Blinn-Phong shading to novel views, as shown in Fig. 2c.

Gooch model. The specular aspect of the Blinn-Phong model (see Fig. 2d) plays a role in more stylistic shading methods as well. The cool-to-warm lighting model introduced by Gooch *et al*. [8] restricts shading to mid-tones, affording greater visibility to specular highlights and edge lines. Here, the diffuse shading is an interpolation between *cool* and *warm* colour tones.

$$i_d = \left(\frac{1 + \hat{\mathbf{L}} \cdot \hat{\mathbf{N}}}{2}\right) k_{warm} + \left(1 - \frac{1 + \hat{\mathbf{L}} \cdot \hat{\mathbf{N}}}{2}\right) k_{cool} \tag{2}$$

where

$$\begin{aligned} k_{cool} &= k_{blue} + \alpha k_d \\ k_{warm} &= k_{yellow} + \beta k_d \end{aligned} \tag{3}$$

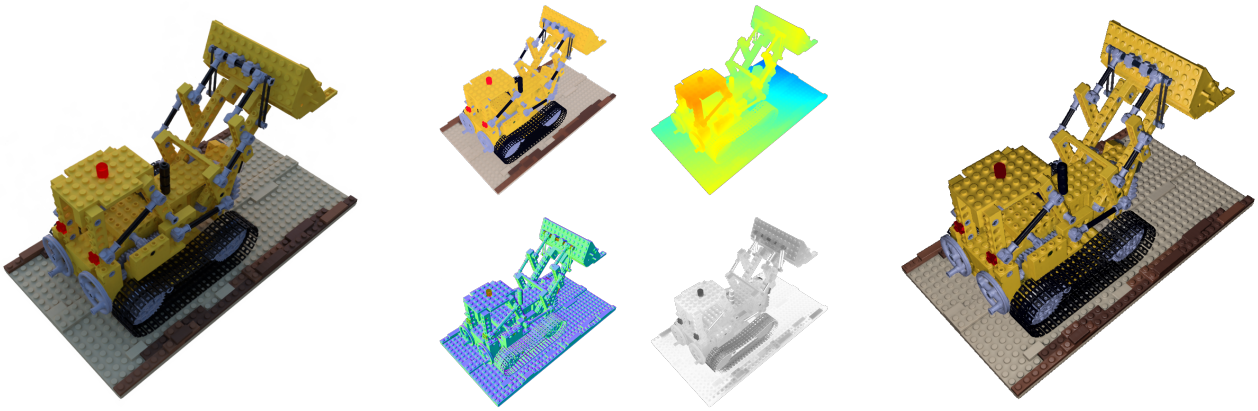
for

$$\begin{aligned} k_{blue} &= (0, 0, b) \\ k_{yellow} &= (y, y, 0) \end{aligned} \tag{4}$$

in RGB space, where b, y, α and β are free parameters.

The resulting effect is one in which the surface points facing towards the light have yellow-orange (warm) hues and those facing away from it are shaded with blue-green (cool) tones. Crucially, the properties involved (k_d, $\hat{\mathbf{L}}$ and $\hat{\mathbf{N}}$) are the same ones required by the diffuse computation of Blinn-Phong, which makes the extension of our approach to Gooch shading on radiance fields trivial (see Fig. 2e).

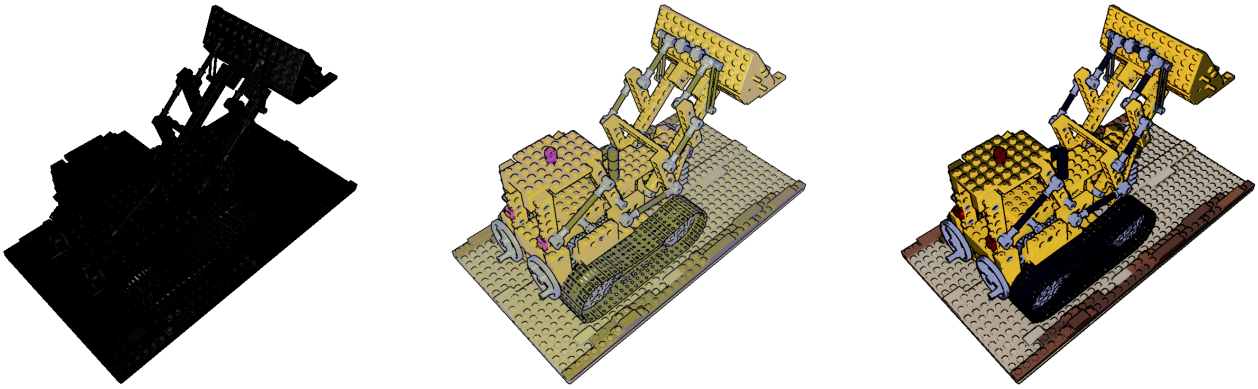
Cartoon shading. We adapt the hard shading technique described by Lake *et al*. [13] to achieve the flat, two-dimensional style prevalent in cartoons and comic books. As opposed to smoothly interpolating the diffuse colour across a surface as in the other models described so far, we define a boundary between shadowed and illuminated surfaces as the point at which the cosine of the incident angle, $\hat{\mathbf{L}} \cdot \hat{\mathbf{N}}$, crosses a given threshold θ.



(a) Standard RGB rendering.

(b) Albedo (top-left), depth (top-right), normal (bottom-left) and roughness (bottom-right) maps recovered with inverse rendering.

(c) Blinn-Phong shading.



(d) The specular component only.

(e) Gooch shading.

(f) Cartoon style.

Figure 2. Different ways of shading the same view. Outlines are drawn on the Gooch and cartoon style renders using the silhouette edge detection algorithm described in Sec. 3.3.

$$i = \begin{cases} k_d k_a & \hat{\mathbf{L}} \cdot \hat{\mathbf{N}} \leq \theta \\ k_d & \hat{\mathbf{L}} \cdot \hat{\mathbf{N}} > \theta \end{cases} \tag{5}$$

Our results are shown in Fig. 2f.

3.2. Image Space Edge Detection

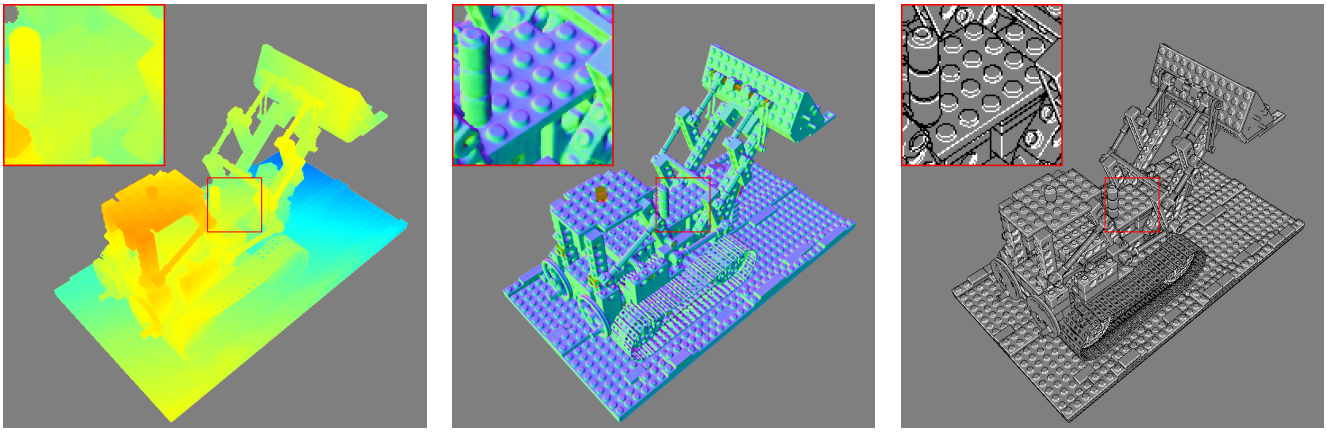
Drawing visible border lines to represent the profile (silhouette) and surface discontinuities (creases) of an object is an important characteristic of the non-photorealistic styles we have been dealing with: technical illustration employing the Gooch model imitates the black exterior and white interior lines conventionally used by illustrators [9], while the bold, black edge lines seen in cartoon and cel animation are recreated in cartoon-style rendering [6].

We can draw edges using image processing techniques, such as the Canny edge detector [2], or the Sobel operator [19]. Although it is possible to perform edge detection directly on the image output, such edges typically do not directly correspond to silhouettes and creases [10]. Instead, we carry out the chosen algorithm on the depth maps (for silhouette edges) and normal maps (for crease lines) of the view.

Our results using Canny edge detection are shown in Fig. 3d. There is a significant drawback here which cannot be shown in the image: there is a lack of cross-view consistency, meaning there are distinct flickering artifacts between similar views as large sections of lines appear and disappear all at once. This effect can be seen in the supplementary videos on our project page.

When using the Sobel operator, we compute the image gradients of the depth and normal maps, and label the points where these exceed a certain threshold as edges. This results in bold outlines with a good degree of cross-view consistency, as seen in Fig. 3e and the supplementary videos.

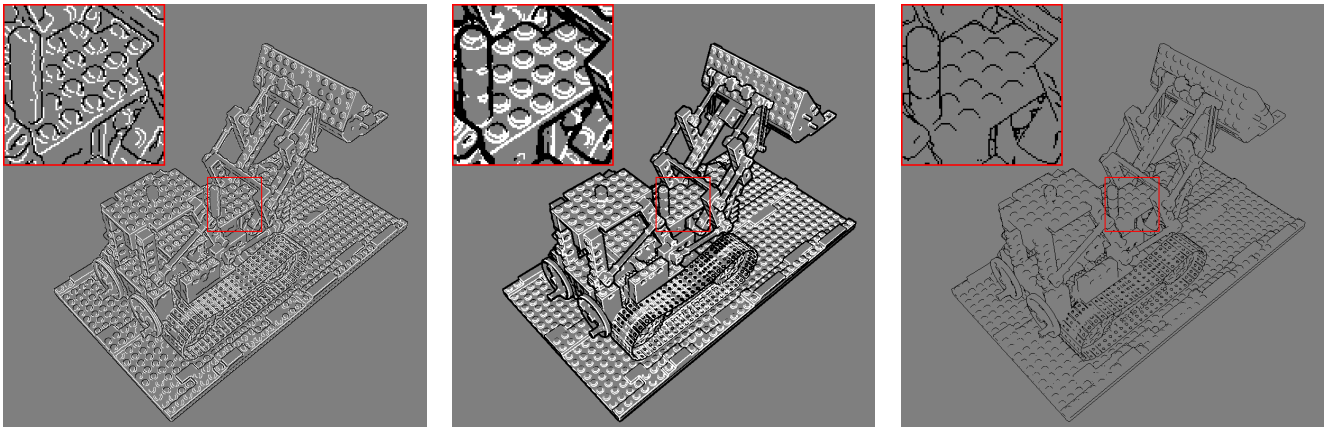
In Fig. 11 of the appendix, we demonstrate how edges can be integrated into stylised rendering. In Fig. 11b, we combine Gooch shading with dark silhouettes and light



(a) The depth map used for silhouette edge detection.

(b) The normal map used for crease line detection.

(c) The lines which we expect to detect, rendered in Blender.



(d) Edge detection using the Canny edge detector.

(e) Edge detection based on the Sobel operator.

(f) Silhouette edge detection using our method.

Figure 3. A comparison of different edge detection methods. Where applicable, silhouette edges are drawn in black, while crease lines are drawn in white.

crease lines to highlight internal edges, while in Fig. 11c, we draw black edges to help discern object shape in cartoon stylisation.

3.3. Novel Silhouette Edge Detection Algorithm

Silhouette edges can be thought of as the intersections between front-facing and a back-facing surfaces, as illustrated in Fig. 5. In mesh representations, a silhouette occurs on the edge between two connected polygons, one of which faces towards the camera, and the other of which faces away from it [9, 10, 13]. Radiance fields do not explicitly define surfaces as polygon meshes do, which prompts us to search for another way of finding where front- and back-facing surfaces meet.

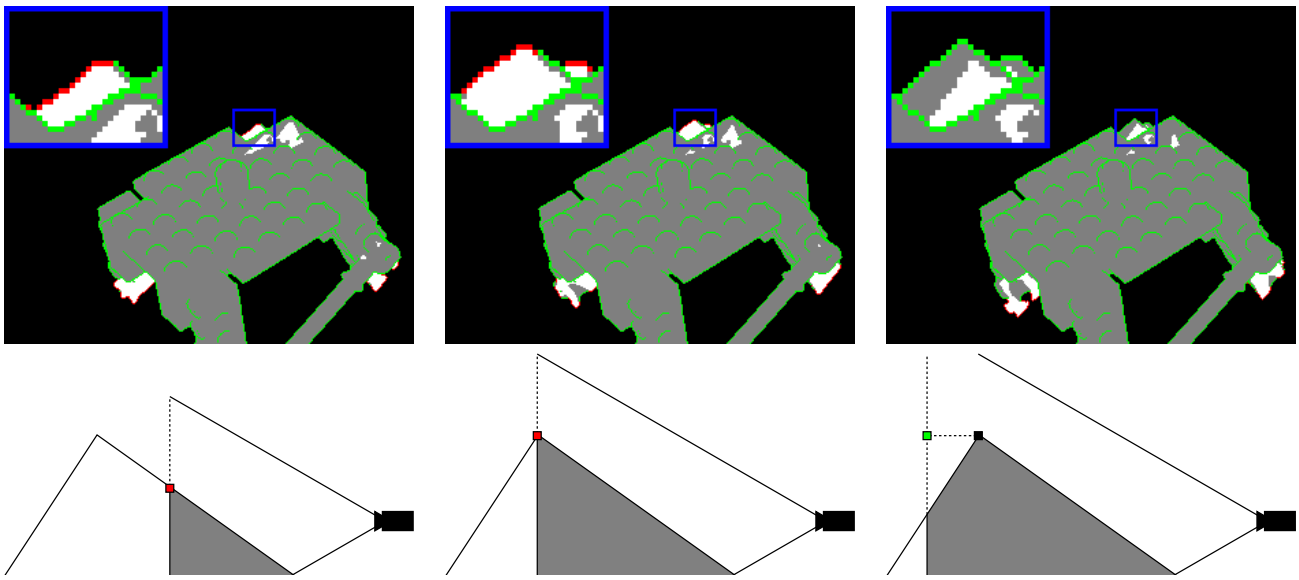
We present an approach that leverages the volumetric nature of radiance fields instead. Volumetric representations can be cross-sectioned to obtain 2D slices at various depths – a property which we use to our advantage by iteratively analysing the volume of an object, one evenly-spaced slice at a time, to infer the shape of its surface.

We demonstrate our process in Fig. 4: each cross-sectional slice of the scene is examined in order of increasing distance from the camera. As a surface comes into view, it appears to expand in successive iterations (see Fig. 4a), with the outer boundary reaching its greatest extent at the silhouette point (see Fig. 4b), before receding in later slices (see Fig. 4c). It is therefore the boundaries of the cross-sectional slices at their greatest extent that define the final silhouette as perceived by the viewer. We handle occlusion by keeping track of which locations have already been covered by previous slices – outlines cannot occur in places which are occluded by a surface that was intersected first.

Our approach produces a 2D map of silhouette points, which we compare with other methods in Fig. 3f. The process can be summarised as follows:

For each 2D slice:

1. Identify and mark all pixels that represent occupied loca-



(a) We track the outer boundary of the surface from the camera's perspective in screen space while scanning through the object.

(b) The silhouette is where front-facing and back-facing surfaces intersect. For the algorithm, this is where the boundary of the active layer reaches its greatest extent.

(c) Beyond the silhouette point, the surface appears to draw away from the boundary. We consider the last marked boundary to be a part of the final silhouette.

Figure 4. An illustration of the process used by our edge-finding algorithm, with some images from its execution, in which the current layer is highlighted in white, previous layers are shown in grey, marked (potential) edges are coloured red and finalised silhouette points are coloured green.

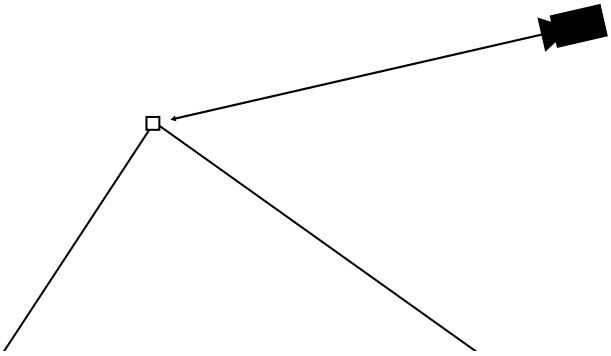


Figure 5. A silhouette point (shown by the white square) is defined as the point where a surface facing towards the camera intersects a surface facing away from the camera.

tions.

2. Label any pixels on the outer boundary of the current slice as potential edges, unless they are already occluded by previous layers.
3. Mark as finalised silhouette points any pixels that were previously labeled as potential edges but are no longer at the boundary in the current slice (i.e., not touching occupied pixels).

Slicing method. Next, we must address the question of how to decide which points make up a slice. In particular, determining the criteria for when a point is considered occupied or unoccupied for the purposes of finding the boundaries. An obvious solution would be to sample the volume density σ, and label the points in which σ exceeds a predetermined threshold σ_{min} as occupied.

$$S_i = \sigma_i \geq \sigma_{\min} \tag{6}$$

where σ_i is the volume density of a particular pixel at the ith iteration of the algorithm and S_i is a predicate determining whether or not the pixel is considered occupied in that iteration (note that each iteration corresponds to a different cross-sectional slice of the scene).

Although this approach works well in most cases, it ignores the reality of how volume density is distributed in a radiance field, where a surface is defined by the cumulative contributions of multiple points along the path of a cast ray. The issue is evident in Fig. 6: a threshold that is too low can lead to obtrusive artifacts due to seemingly random low volume density patches left over from training the radiance field, while choosing a value of $\sigma_{\min}$ that is too high results in holes in the surface in spots where σ_i does not reach the threshold in any particular slice, even if these areas appear solid in the rendered radiance field.

While we can generally get around the limits of the aforementioned method by choosing a suitable value of $\sigma_{\min}$, we favour an alternative that tracks the accumulated

$\sigma_{\min} = 0.01$ $\sigma_{\min} = 1$ $\sigma_{\min} = 10$ $\sigma_{\min} = 25$ $\sigma_{\min} = 50$

Figure 6. The effects on our algorithm of changing the threshold $\sigma_{\min}$ using the σ-based slicing method, where a point is considered occupied if $\sigma_i \geq \sigma_{\min}$. The layers which have been passed through by the algorithm are coloured grey. Setting the threshold too low causes artifacts to appear in the foreground, while setting the threshold too high can cause the algorithm to skip low density areas.

$A_{\min} = 0.01$, $\alpha_{\min} = 0.25$ | $A_{\min} = 0.8$, $\alpha_{\min} = 0.0001$ | $A_{\min} = 0.8$, $\alpha_{\min} = 0.25$ | $A_{\min} = 0.8$, $\alpha_{\min} = 0.5$ | $A_{\min} = 1$, $\alpha_{\min} = 0.25$

Figure 7. The alternative slicing method in Eq. (10), which uses a combination of α and accumulated opacity A to determine occupancy. We can still "break" the process by setting the opacity threshold $A_{\min}$ too high or too low – but once a suitable value is found, we only need to modulate $\alpha_{\min}$ to vary the level of detail.

opacity A_i of each pixel:

$$\begin{aligned} A_0 &= 0 \\ A_i &= A_{i-1} + \alpha_i(1 - A_{i-1}) \end{aligned} \tag{7}$$

Here, A_i recursively depends on the accumulated opacity of the pixel with the same position in the last iteration, A_{i-1}, and

$$\alpha_i = 1 - \exp\left(-\sigma_i \delta_i\right) \tag{8}$$

where δ_i is the distance between samples.

In order for a pixel in a given slice to be considered occupied with this approach, the accumulated opacity (which depends on preceding iterations as well as the actual one) must exceed a threshold $A_{\min}$. This allows us to avoid the possibility of artifacts appearing in low density regions as we saw in the other approach, since there is not enough of an accumulation of opacity in these patches for it to surpass the threshold. To avoid the appearance of holes, we always mark a point as occupied when it is the first in its path to have an accumulated opacity greater than the threshold, $A_i \geq A_{\min}$. We mark subsequent points in the same path as occupied only if they are a significant contributor to the accumulated opacity:

$$\alpha_i \geq \alpha_{\min} \cdot A_i \tag{9}$$

where $\alpha_{\min}$ is a predetermined threshold.

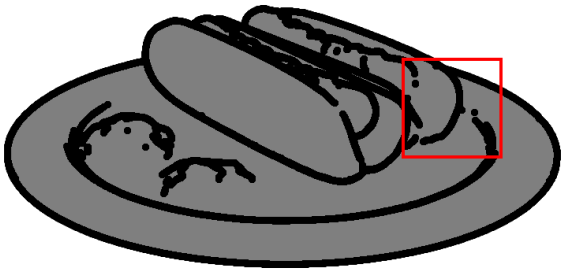
Overall, the predicate S_i determining the occupancy of a pixel using this method is as follows:

$$S_i = \begin{cases} A_i \geq A_{\min} & \text{if } S_j \text{ is false for all } j < i \\ \alpha_i \geq \alpha_{\min} \cdot A_i & \text{if } S_{i-1} \text{ is true} \\ \text{false} & \text{otherwise} \end{cases} \tag{10}$$

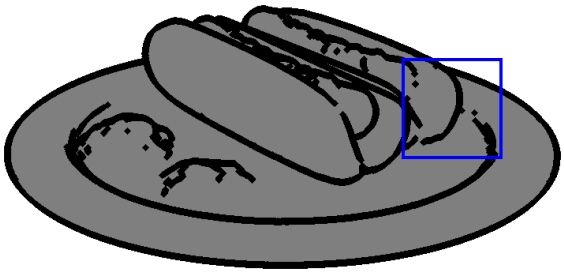
As demonstrated in Fig. 7, this method of checking occupancy does not produce holes or artifacts as the value of $\alpha_{\min}$ is varied, but instead results in a loss of detail for lower values, and a tendency to produce false positives in areas with sharp gradients for high values.

Line weight. So far, we have discussed an algorithm that produces a one-pixel-thick outline map composed of silhouette points. We may wish to draw lines which are more than a single pixel thick, however, to enhance visibility or emphasise depth.

Extending the algorithm to encode depth is trivial: we just need to store the depths of the slices where each silhouette point is first encountered. To render the points with varying thickness based on their proximity to the camera, we assign weights using linear interpolation:



(a) Euclidean distance approach from Eq. (12).



(b) Manhattan distance approach from Eq. (13).

(c) A close-up view illustrating the contrasting stroke styles. Euclidean distance creates smooth, circular strokes (left), while Manhattan distance produces sharp, blocky edges (right).

Figure 8. Heavily exaggerated lines produced by assigning all silhouette points a constant weight (with $a = b = 5$), and then increasing their thickness using either the Euclidean distance formula (Eq. (12)) or the Manhattan distance formula (Eq. (13)). We observe how the choice of approach gives different line styles.

$$w(d) = \frac{(d - d_{\min})(b - a)}{d_{\max} - d_{\min}} + a \tag{11}$$

Here, d represents the depth of the silhouette point, and $d_{\min}$ and $d_{\max}$ are the lowest and highest possible depth values, respectively. The line weights to be interpolated between are denoted by the free parameters a and b – a higher weight corresponds to a greater line thickness. It is worth noting that $w(d_{\min}) = a$ and $w(d_{\max}) = b$. If a constant line thickness is desired, a and b can be set to the same value.

To draw thicker outlines, we then place appropriately sized spots at the weighted silhouette points, using larger spots for greater weights. We use the following formula to compute the contribution of silhouette point c at coordinates (c_x, c_y) to the final intensity value of the pixel at (x, y):

$$I_c = \max\left(w_c - \sqrt{(x - c_x)^2 + (y - c_y)^2}, 0\right) \tag{12}$$

where w_c is the weight of the point c. While we have assigned an intensity value proportional to weight and Euclidean distance in this formula, alternatives such as Manhattan distance are also possible:

$$I_c = \max\left(w_c - |x - c_x| - |y - c_y|, 0\right) \tag{13}$$

The choice of formula subtly affects the style of lines produced by the algorithm, as shown in Fig. 8. Regardless of the method used to determine the individual contributions I_c of each silhouette point, the final intensity at a pixel is obtained by summing these contributions, clamped to a maximum value of 1.

$$I_{\text{final}} = \min\left(\sum_c I_c, 1\right) \tag{14}$$

We include more results showing the effects of changing the weight parameters a and b, and the resulting thickened lines, in Fig. 12 of the appendix.

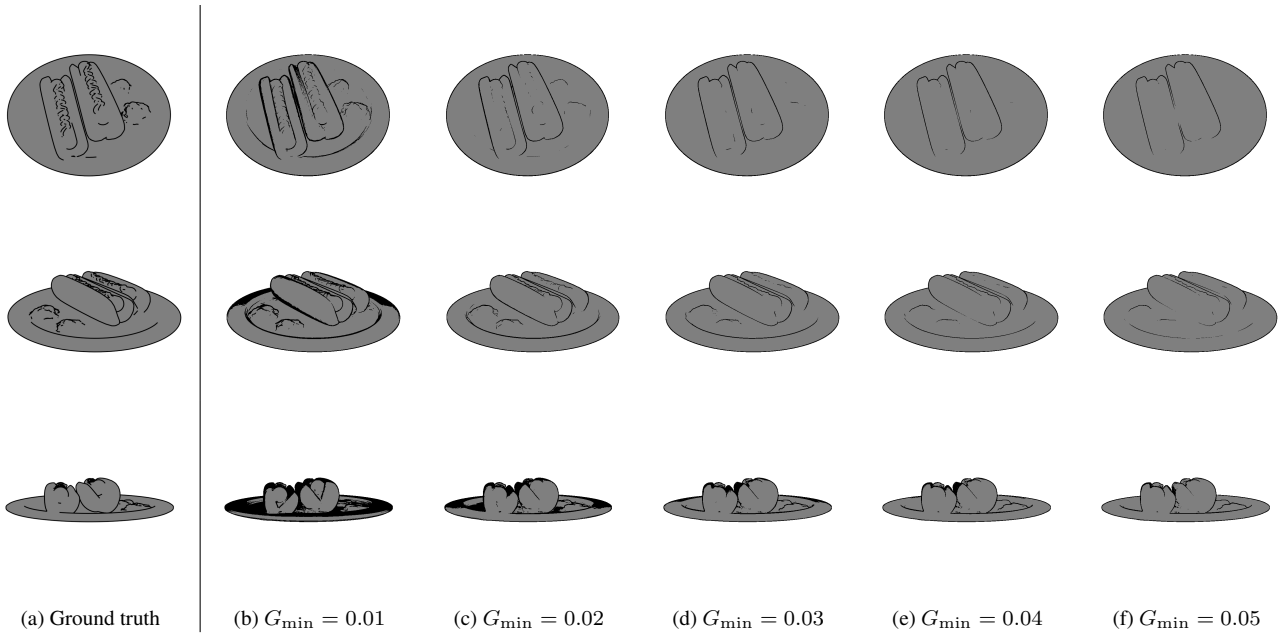
4. Results

A selection of scenes rendered using our stylisation methods is shown in Fig. 1. This is a highly subjective domain, particularly on the topic of shading, so in this section we focus primarily on the differences between edge detection methods and the impacts of the various parameters and optimisations introduced in the previous section.

Setup. All results pictured in this paper are obtained using our methods implemented within TensoIR [11], a framework for inverse rendering. TensoIR is built upon TensoRF [3], which provides a framework for modelling radiance fields as explicit voxel feature grids.

Our dataset comprises two synthetic scenes (the Lego model and hot dog) from the NeRF dataset [14], one synthetic scene (the armadillo) from the Stanford 3D scanning repository [4], and a real-world scene (the Ignatius statue) from the *Tanks and Temples* dataset [12].

Comparison of silhouette edge detection methods. We now evaluate a difficult case for the silhouette edge detection algorithms discussed in Secs. 3.2 and 3.3. We focus on



(a) Ground truth (b) $G_{\min} = 0.01$ (c) $G_{\min} = 0.02$ (d) $G_{\min} = 0.03$ (e) $G_{\min} = 0.04$ (f) $G_{\min} = 0.05$

Figure 9. The effects of the choice of threshold $G_{\min}$ when using the Sobel operator for silhouette edge detection. The expected results, rendered in Blender, are shown in the leftmost column.

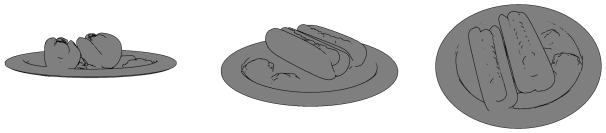


Figure 10. The edges drawn by our algorithm using opacity-based slicing with the parameters $A_{\min} = 0.6$, $\alpha_{\min} = 0.1$. With the appropriately selected parameters, all views maintain a high level of detail without false positives at high depth gradients.

the Sobel operator and our volumetric edge detection algorithm, and how their parameters can be tuned to achieve the desired effects.

In Fig. 9, we show how using different thresholds on the Sobel operator affects the resulting outlines. We notice how lower thresholds produce more false positives on surfaces that make a sharp angle with the camera (see Fig. 9b) due to the large, continuous depth changes in these areas. While choosing a higher threshold does alleviate this, it also results in a great loss of detail on other views which may not have such sharp depth gradients (see Fig. 9f).

We performed the same procedure on our own algorithm, the full results of which are shown in Figs. 13 and 14 in the appendix. A selected result from Fig. 14c is shown in Fig. 10, where we see a constant level of detail across all views without false positives in continuous regions.

5. Conclusion

In this paper, we have demonstrated how radiance fields are a useful medium not just for faithful reconstruction of real-life scenes, but also for abstract, artistic or technical stylisation. We build upon the contributions of inverse rendering to enable non-photorealistic shading styles, and draw silhouettes and creases to emphasise shape and geometry. We also contribute a novel algorithm for detecting silhouette edges in radiance fields, suitable for finding and enhancing the outlines of an object.

6. Responsible Research

In the interest of ensuring the reproducibility of our results, and to allow scrutiny of our implementations, our code is made publicly available at `https://github.com/mszilvasy/TensoIR-NPR`. As mentioned, the datasets we use are all available in the public domain [4, 12, 14], and the code repository also includes the config files needed to train and render the scenes presented in this paper.

References

[1] James F. Blinn. Models of light reflection for computer synthesized pictures. *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, 1977. 2

[2] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, 1986. 3

[3] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 1, 7

[4] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 7, 8

[5] Cassidy Curtis, Sean Anderson, Joshua Seims, Kurt Fleischer, and David Salesin. Computer-generated watercolor. *Proc. SIGGRAPH1997*, 97, 06 1997. 1, 2

[6] Philippe Decaudin. Cartoon-looking rendering of 3d-scenes. 2003. 1, 3

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016. 1

[8] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 447–452, New York, NY, USA, 1998. Association for Computing Machinery. 1, 2

[9] Bruce Gooch, Peter-Pike J. Sloan, Amy Ashurst Gooch, Peter Shirley, and Richard F. Riesenfeld. Interactive technical illustration. In *SI3D*, 1999. 1, 2, 3, 4

[10] Aaron Hertzmann. Introduction to 3d non-photorealistic rendering: Silhouettes and outlines. 1999. 1, 3, 4

[11] Haian Jin, Isabella Liu, Peijia Xu, Xiaoshuai Zhang, Songfang Han, Sai Bi, Xiaowei Zhou, Zexiang Xu, and Hao Su. Tensoir: Tensorial inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1, 2, 7

[12] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 7, 8

[13] Adam T. Lake, Carl S. Marshall, Mark J. Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *International Symposium on Non-Photorealistic Animation and Rendering*, 2000. 1, 2, 4

[14] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 7, 8

[15] Thu Nguyen-Phuoc, Feng Liu, and Lei Xiao. Snerf: Stylized neural implicit representations for 3d scenes, 2022. 1

[16] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18:311 – 317, 1975. 2

[17] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990. 1, 2

[18] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 1

[19] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014. 3

[20] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. SIGGRAPH '94, page 91–100, New York, NY, USA, 1994. Association for Computing Machinery. 1, 2

[21] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields, 2022. 1

[22] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *ArXiv*, abs/2010.07492, 2020. 1

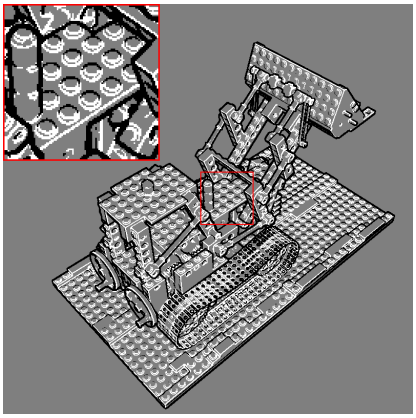
[23] Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul E. Debevec, William T. Freeman, and Jonathan T. Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM Trans. Graph.*, 40:237:1–237:18, 2021. 1, 2

[24] Yuechen Zhang, Zexin He, Jinbo Xing, Xufeng Yao, and Jiaya Jia. Ref-npr: Reference-based non-photorealistic radiance fields for controllable scene stylization, 2023. 1

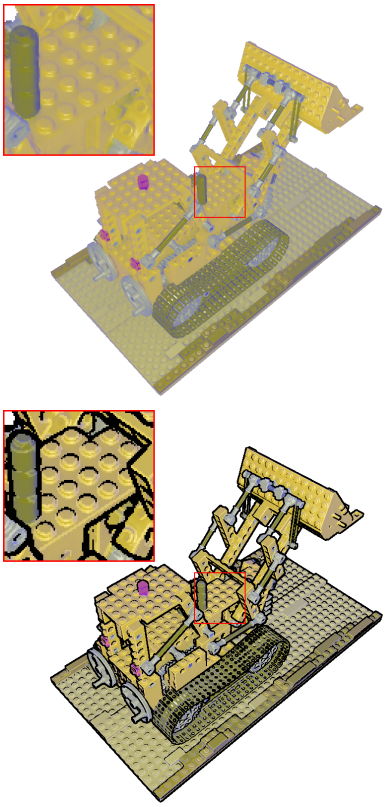
[25] Yuanqing Zhang, Jiaming Sun, Xingyi He, Huan Fu, Rongfei Jia, and Xiaowei Zhou. Modeling indirect illumination for inverse rendering. In *CVPR*, 2022. 1, 2

A. Appendix

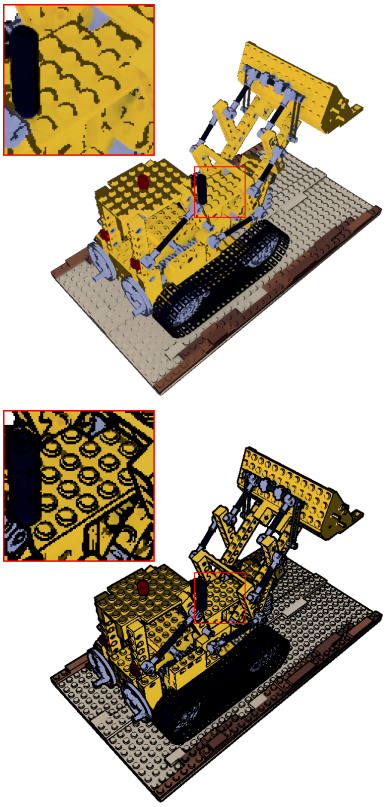
The following pages include all figures which could not be incorporated into the main body of the document.



(a) Silhouette edges (black) and crease lines (white) detected using the Sobel operator. This is the same view that was shown in Fig. 9.

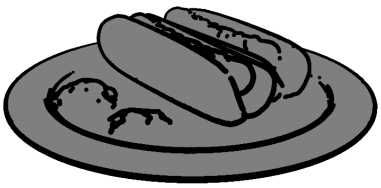


(b) Gooch shading (top) with dark silhouette edges and light crease lines (bottom).

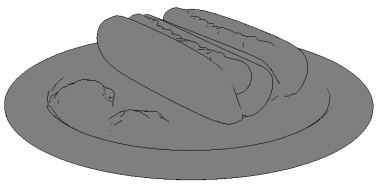


(c) Toon shading (top) with silhouette edges and crease lines drawn in black (bottom).

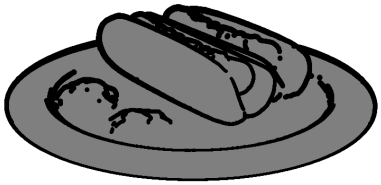
Figure 11. Stylised rendering with edges derived using the Sobel operator.



(a) $a = 8, b = 0$

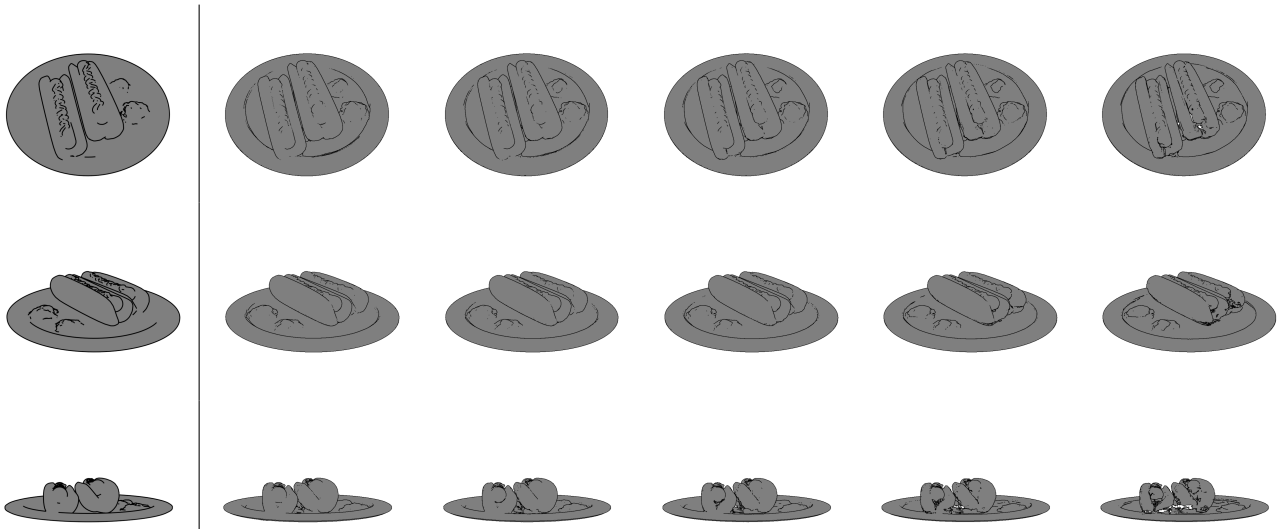


(b) $a = 1, b = 1$



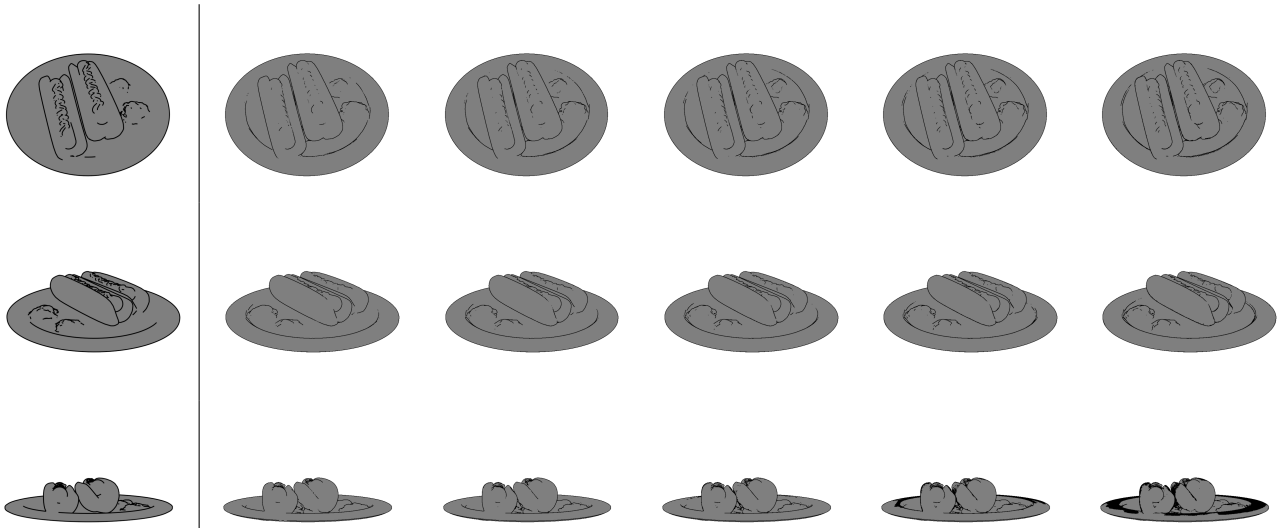
(c) $a = 1, b = 9$

Figure 12. Exaggerated lines showing how changing the weight parameters a and b in Eq. (11) lets us vary line thickness. Increasing the value of a results in heavier lines close to the camera, while higher values of b yield thicker lines further away.



(a) Ground truth (b) $\sigma_{\min} = 2$ (c) $\sigma_{\min} = 4$ (d) $\sigma_{\min} = 6$ (e) $\sigma_{\min} = 8$ (f) $\sigma_{\min} = 10$

Figure 13. The effects of the choice of threshold $\sigma_{\min}$ when using our silhouette detection method with σ-based slicing in Eq. (6). We observe how higher thresholds cause low density regions to be missed by the algorithm.



(a) Ground truth (b) $\alpha_{\min} = 0.05$ (c) $\alpha_{\min} = 0.1$ (d) $\alpha_{\min} = 0.15$ (e) $\alpha_{\min} = 0.2$ (f) $\alpha_{\min} = 0.25$

Figure 14. The silhouettes detected for the same views using the opacity-based slicing method in Eq. (10). The accumulated opacity threshold is fixed at $A_{\min} = 0.6$ for all columns, and only the the threshold $\alpha_{\min}$ is varied.