Physically-based Feature Line Rendering

REX WEST, The University of Tokyo, Japan

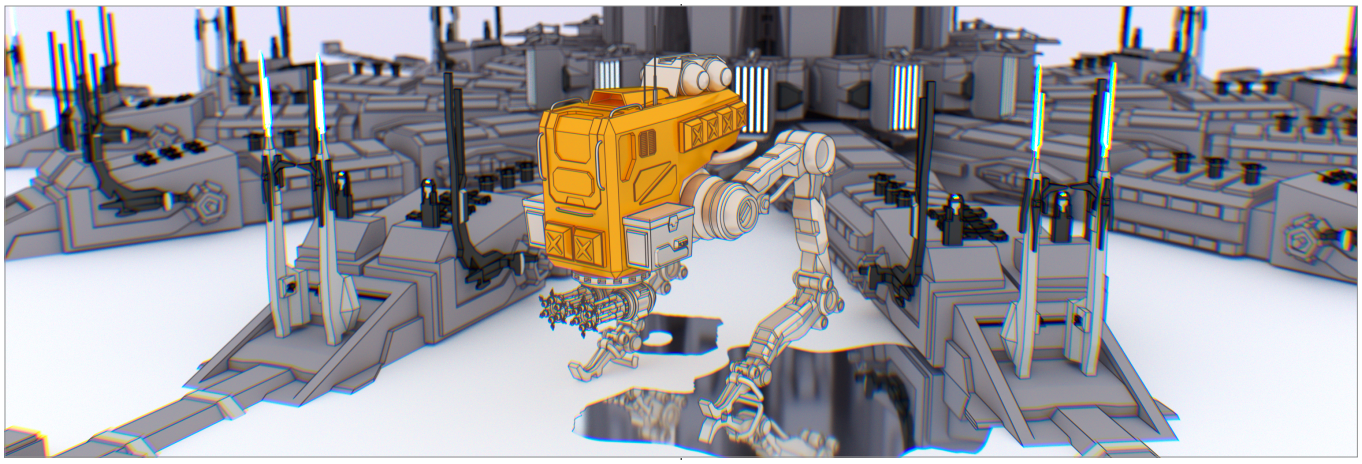

Fig. 1. Our method makes it possible to render feature lines in the presence of physical effects such as depth-of-field blur, glossy reflections, and dispersion. This scene was rendered with a spectral renderer, and demonstrates the robustness of our method even in physically complex rendering scenarios.

Feature lines visualize the shape and structure of 3D objects, and are an essential component of many non-photorealistic rendering styles. Existing feature line rendering methods, however, are only able to render feature lines in limited contexts, such as on immediately visible surfaces or in specular reflections. We present a novel, path-based method for feature line rendering that allows for the accurate rendering of feature lines in the presence of complex physical phenomena such as glossy reflection, depth-of-field, and dispersion. Our key insight is that feature lines can be modeled as view-dependent light sources. These light sources can be sampled as a part of *ordinary paths*, and seamlessly integrate into existing physically-based rendering methods. We illustrate the effectiveness of our method in several real-world rendering scenarios with a variety of different physical phenomena.

CCS Concepts: • **Computing methodologies → Rendering**; **Ray tracing**.

Additional Key Words and Phrases: rendering, non-photorealistic rendering, feature lines

ACM Reference Format:
Rex West. 2021. Physically-based Feature Line Rendering. *ACM Trans. Graph.* 40, 6, Article ? (December 2021), 11 pages. https://doi.org/10.1145/3478513.3480550

Author's address: Rex West, The University of Tokyo, Japan.

1 INTRODUCTION

Feature lines are an important part of many Non-photorealistic Rendering (NPR) styles, such as those used in comic book illustrations, animated films, and technical drawings. They help visualize the shape and form of objects, and can be used to emphasize important geometric details. More technically, feature lines are a view-dependent visualization of the geometric properties of 3D objects such as contours, edges, creases, and discontinuities.

There is an extensive amount of existing work on feature line rendering [Bénard and Hertzmann 2019] that generally falls into two categories: object-space methods and screen-space methods. Object-space methods work directly on properties of the objects, such as surface curvature, and their computational cost scales proportionally to geometric complexity. Screen-space methods operate on intermediate screen buffers, or even directly on final rendered images, and scale proportionally to the pixel resolution, making them efficient for geometrically complex scenes.

One particular subset of screen-space feature line rendering methods, those based on ray casting, are particularly attractive for their rendering quality, sub-linear scaling proportional to geometric complexity, and ease of integration into modern rendering workflows. Unfortunately, existing ray-based methods are limited to rendering feature lines for immediately visible surfaces and specular reflection and refraction, and have difficulty rendering feature lines in the presence of physical effects such as glossy reflections, depth-of-field, and dispersion.

Physically-based Rendering (PBR), on the other hand, handles such effects as a natural consequence of a physically correct formulation. PBR simulates how light behaves in the real-world by considering all of the different paths light can travel between a virtual camera and light sources [Veach 1997]. Each path captures

the different emission, scattering, and absorption events that make up a wide variety of physical effects, and, conversely, effects that can be modeled as such events can be naturally incorporated into PBR methods.

We propose a novel path-based method that merges feature line rendering and PBR by modeling feature lines as view-dependent, implicit light sources. We observed that existing ray-based methods, when viewed in the context of paths, essentially treat feature lines as a light source. By treating feature lines as a light source, and testing for intersection at each edge of a path, we can construct *ordinary* paths that preserve the look-and-feel of existing methods while also supporting new physical effects. Furthermore, since they are ordinary paths, we can directly leverage the capabilities of existing path-based PBR methods, and integrate feature line rendering seamlessly into modern renderers [Pharr et al. 2018].

We implement our method on a spectral renderer, and empirically demonstrate how our method plausibly renders feature lines in the presence of several physical effects including glossy reflections, depth-of-field, and spectral dispersion. We compare our method to the existing ray-based methods of Choudhury and Parker [2009]; Ogaki and Georgiev [2018], and additionally demonstrate how various aspects of our method can be controlled artistically.

2 BACKGROUND AND RELATED WORK

In NPR, we are interested in generating images that contain expressive visuals and effects. These effects are often artistic in nature and generally do not follow the laws of light physics. One such effect is feature lines, which uses 2D line primitives to express the shape and form of 3D objects. Visualizing these lines plays an important role in certain types of visualizations and artistic media such as comic books and animated films.

2.1 Feature lines

Feature lines can be drawn to express the silhouette of an object and sharp changes in surface curvature [Saito and Takahashi 1990; DeCarlo et al. 2003, 2004], apparent ridges and valleys in the surface [Ohtake et al. 2004; Judd et al. 2007], and even sharp changes in lighting [Lee et al. 2007]. They, in general, represent extremal values or discontinuities in some measurable quantity of an object. There are two general categories of feature line rendering methods: object-space and screen-space.

Object-space methods work directly on object geometry. DeCarlo et al. [2003, 2004] introduce an object-space method (and an image-space variant) for visualizing shape-suggesting contours by placing feature lines at the locations where the radial curvature of the surface geometry is zero. Similarly, Ohtake et al. [2004] visualize ridges and valleys in object geometry by fitting an implicit surface and estimating the first and second order curvature derivatives, and Judd et al. [2007] visualize apparent ridges of an object by computing the view-dependent curvature and placing feature lines at the locations where it is maximized. While effective, object-space methods can become computationally expensive as the geometric complexity increases.

On the other hand, screen-space methods work directly on an image or a screen-space buffer, and their computational cost scales with the number of pixels, independent of the complexity of the geometry. Saito and Takahashi [1990] highlight the geometric qualities of a rendered image by placing feature lines along edges detected in a screen-space buffer. Lee et al. [2007] visualize contours of an object by rendering a tone image and placing feature lines along the boundaries of shaded regions. Rather than storing an explicit screen-space buffer, Choudhury and Parker [2009] present a screen-space post-process that uses ray stencils, a variant of cone tracing, to estimate if a contour falls within each pixel. Ogaki and Georgiev [2018] extend the method of Choudhury and Parker [2009] to reuse cached shading ray samples for contour detection and support specular reflection and refraction.

Several other works address challenges in feature line rendering including, specular reflection and refraction [Kim et al. 2008], temporal flickering in animation [Bauer 2017], and stereo-consistency [Bukenberger et al. 2018].

There has also been research into how humans perceive line drawings and why artists place lines where they do. Cole et al. [2008] and Hertzmann [2020, 2021] present insight on the human perception of line drawings and their relation to feature line rendering. Cole et al. [2008] suggest that existing feature line methods place lines in locations similar to human artists, but that there are still discrepancies. In contrast, the more recent work of Hertzmann [2021] suggests that existing methods that place feature lines at edges or discontinuities best estimate where people would draw lines.

Bridging the gap between line drawings and feature line rendering, Liu et al. [2020] present a feature line rendering method that uses a machine learning based approach to learn where to place feature lines. Their method maps geometric and view-based data of a 3D model to line drawings, and provides a promising direction for producing feature line renderings that are closer to what an artist would draw.

For interested readers, Bénard and Hertzmann [2019]; DeCarlo [2012] provide thorough surveys of the history and prior work on contour visualization and feature line rendering.

2.2 Ray-based feature line rendering

Of the many feature line rendering methods, ray-based methods [Choudhury and Parker 2009; Ogaki and Georgiev 2018] are particularly attractive. Their computational cost scales sub-linearly with the complexity of the geometry, making them effective for feature line rendering in the highly complex scenes used in production rendering. Many modern production renderers [Pharr et al. 2018] are based on ray and path sampling, allowing ray-based feature line methods to integrate naturally into modern production workflows.

In general, ray-based methods use ray-object intersection tests to determine where to draw feature lines. However, many of the contours we would like to visualize are infinitesimally thin, and rays are unable to directly intersect them. Instead, we can approximate their location by comparing a query ray that we want to test line intersection for, and one or more sample rays (see Fig. 3). If certain properties of the query ray's hit point and a sample ray's hit point are sufficiently different, there is likely a sharp change in value or a discontinuity between them.

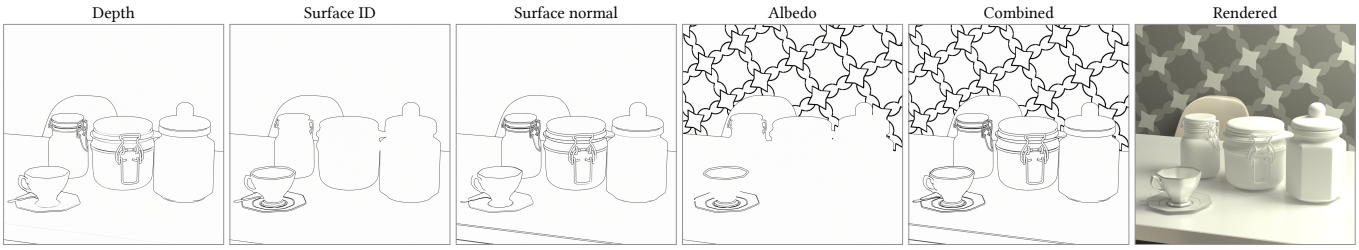

Fig. 2. Various metrics can be used to draw feature lines that emphasize geometric and artistic properties of the scene. Here we demonstrate how depth, surface ID, surface normal, and material albedo affect the lines drawn, and show how they can be combined to produce a more complete rendering that captures the essence of the rendered image on the far right.

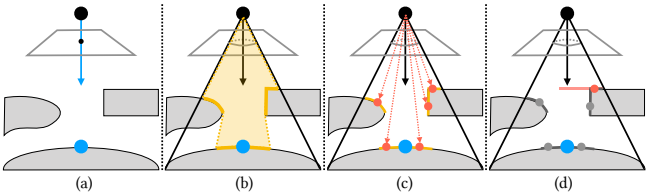

Fig. 3. To determine if a query ray (a, blue) intersects a feature line, we compare its hit point (b, blue) to the surface encapsulated in a region (b, yellow). Computing the exact encapsulated region and comparing against all points within that region is often infeasible in practice, so we make an approximation using a finite number of ray samples (c, red). The samples are then compared to the query ray's hit point (c, blue), and the closest sample with a feature line (d, red line) that intersects the ray is chosen.

In practice we can measure different types of metrics that help identify discontinuities we want to visualize (see Fig. 2). For example, discontinuities in surface ID and depth of the ray samples can suggest the outline and intersections of objects, and sharp changes in surface normal can suggest the presence of creases and folds in a surface. For a perfectly accurate solution, we would want to test the query ray against all rays within a line-width region of screen-space, but we can make a reasonable approximation with a finite number of samples (see Fig. 3). If several sample rays have sufficiently different properties, their hit points can be projected onto the query ray and sorted to determine the closest.

Choudhury and Parker [2009] introduced the first ray-based method, and use a variant of cone tracing, called ray stencils, to detect feature lines. Their method projects a screen-space stencil the size of the desired feature lines into 3D space, creating a cone, and this cone encapsulates the region of surface they test for feature discontinuities. Their method, however, only considers a single, uniform line width and surfaces directly visible from the camera (see Fig. 4). The method works well, but the fixed structure of the ray-stencil can introduce aliasing, as well as artifacts in locations where several contours meet. Additionally, the ray sampling required for the stencil can be computationally expensive, and its fixed shape makes it difficult to re-use samples from other nearby queries.

Ogaki and Georgiev [2018] expand upon the work of Choudhury and Parker [2009] to include stochastically drawn ray samples, variable line width and color, and feature lines for specular reflection and refraction (see Fig. 4). A key insight is that modern path-based renderers sample a large number of paths for each pixel, and these paths can be stored in a screen-space cache and re-used for feature line detection. Re-using the first edge of cached path samples is essentially equivalent to using stochastic ray samples, and avoids the overhead of explicitly sampling them. Cached paths can additionally be leveraged for computing feature lines seen through specular reflection and refraction by comparing specular bounces at the same path depth.

Unfortunately, existing ray-based methods are unable to render feature lines seen through physical effects like glossy reflections or depth-of-field, and, in some cases, can produce strong artifacts when such effects are present. These effects should result in *blurry feature lines*, but as existing methods operate as a post-process in screen-space, they incorrectly try to detect *hard feature lines* on *blurry data*.

2.3 Physically-based rendering

In contrast, physically-based rendering methods handle effects like glossy reflections and depth-of-field naturally. They simulate how light behaves in the real world and are capable of generating photo-realistic images for a virtual scene by calculating how much light leaves light sources, bounces around the geometry, and reaches a camera.

This process is formulated as an integral over all of the paths that light could have traveled [Veach 1997],

$$I = \int_{\mathcal{P}} f(\overline{\mathbf{x}}) \mathrm{d}\overline{\mathbf{x}}, \tag{1}$$

where $\overline{\mathbf{x}}$ is a path, $\mathcal{P}$ is the space of all possible paths, and f is the path contribution function. A path of length k is a sequence of $k+1$ vertices and k edges that connect those vertices, $\overline{\mathbf{x}} = [\mathbf{x}_0, \cdots, \mathbf{x}_k]$, and the path contribution $f(\overline{\mathbf{x}})$ is the product of edge-local transport η and vertex-local interactions ρ (see Fig. 5),

$$f(\overline{\mathbf{x}}) = \prod_{i=1}^{k} \eta(\mathbf{x}_{i-1}, \mathbf{x}_i) \prod_{i=0}^{k} \rho(\mathbf{x}_i)\,. \tag{2}$$

In surface rendering, the edge transport $\eta(\mathbf{x}_{i-1}, \mathbf{x}_i)$ describes how light transmits down the edge between two vertices $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$, and expands to a geometry term G,

$$\eta(\mathbf{x}, \mathbf{y}) = G(\mathbf{x}, \mathbf{y}) = V(\mathbf{x}, \mathbf{y}) \frac{D(\mathbf{x}, \mathbf{y}) D(\mathbf{y}, \mathbf{x})}{\|\mathbf{y} - \mathbf{x}\|^2}, \tag{3}$$

where the visibility term $V(\mathbf{x}, \mathbf{y})$ is 1 when there is visibility between $\mathbf{x}$ and $\mathbf{y}$ and 0 otherwise, $D(\mathbf{x}, \mathbf{y})$ is $|\mathbf{n}_\mathbf{x} \cdot \omega_{\mathbf{xy}}|$ when on a surface, $\mathbf{n}_\mathbf{x}$ is the surface normal at $\mathbf{x}$, and $\omega_{\mathbf{xy}}$ is the unit direction vector from $\mathbf{x}$ towards $\mathbf{y}$.

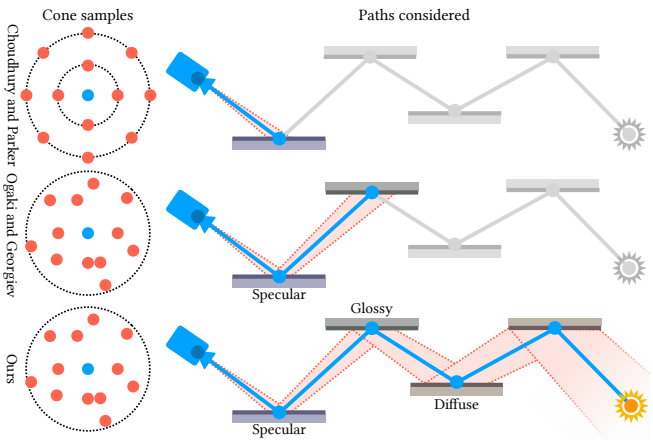

Fig. 4. A comparison of how Choudhury and Parker [2009], Ogaki and Georgiev [2018], and our method generate samples for feature line intersection tests and the different types of paths considered. Choudhury and Parker [2009] use a fixed stencil for generating samples and only consider the first edge of paths. Ogaki and Georgiev [2018] use stochastically drawn samples and additionally consider paths with specular prefixes. Our method also uses stochastic sampling, but extends the feature line intersection test to every edge of arbitrary paths, providing support for all paths in path-space.

Similarly, the vertex interaction $\rho(\mathbf{x}_i)$ describes how light is scattered at a vertex $\mathbf{x}_i$,

$$\rho(\mathbf{x}_i) = \begin{cases} L_e(\mathbf{x}_{k-1}, \mathbf{x}_k) & \text{if } i = k \\ W(\mathbf{x}_0, \mathbf{x}_1) & \text{if } i = 0 \\ \rho_s(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) & \text{if } \mathbf{x}_i \text{ is on a surface} \end{cases}, \tag{4}$$

where $L_e(\mathbf{x}_{k-1}, \mathbf{x}_k)$ is the emission term, $W(\mathbf{x}_0, \mathbf{x}_1)$ is the sensor responsivity term, and ρ_s is the bidirectional scattering distribution function.

Directly computing the analytical solution to Eq. (1) is infeasible in practice, so we instead sample some finite number n of paths $\overline{\mathbf{x}} \sim p$ and use Monte Carlo integration to make an approximation,

$$\int_{\mathcal{P}} f(\overline{\mathbf{x}}) \mathrm{d}\overline{\mathbf{x}} = \mathrm{E}\left[\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})}\right] \approx \frac{1}{n} \sum_{i=1}^{n} \frac{f(\overline{\mathbf{x}}_i)}{p(\overline{\mathbf{x}}_i)}, \tag{5}$$

and as the number n of path estimates $\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})}$ increases, the approximation converges to the true solution.

Readers interested in a more thorough introduction to physically-based rendering can refer to Pharr et al. [2016].

3 PATH-BASED FEATURE LINE RENDERING

The path-based formulation used in physically based rendering naturally handles physical phenomena like glossy reflections and depth-of-field. By formulating feature line rendering in the framework of paths we can leverage the path integral to render feature lines in the presence of these effects.

We first consider how to achieve the behavior of existing methods in the context of PBR. This leads us to two key observations that provide insight on how to seamlessly incorporate feature lines into a path-based formulation.

Observation 1: When considered in the context of paths, existing methods treat feature lines as a light source.

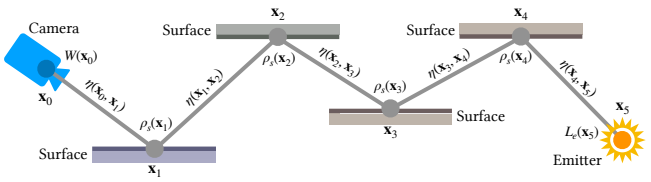

Fig. 5. The terms of the path contribution function Eq. (2) expanded onto a path light could have travelled.

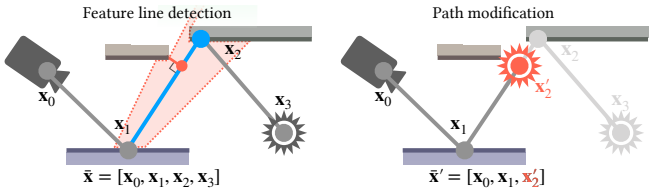

Fig. 6. Each edge of a path is tested for feature line intersection. If an intersection is found, the path is modified to place an emission vertex at the location of the intersection. The throughput and probability density of the modified path can then be computed to determine its contribution to the rendered image.

Existing ray-based methods effectively consider a limited subset of all possible paths. Choudhury and Parker [2009] consider paths with a single edge, and Ogaki and Georgiev [2018] additionally consider paths with specular prefixes[1] (see Fig. 4).

If we consider these two subsets of paths we see that their path sample contribution reduces to,

$$\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})} = L_e(\mathbf{x}_0, \mathbf{x}_1) \qquad \text{and} \qquad \frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})} = \prod_{i=1}^{k-1} \alpha_i L_e(\mathbf{x}_{k-1}, \mathbf{x}_k), \tag{6}$$

for a single edge and specular prefixes respectively (see Appendix B for complete assumptions and derivations).

When a feature line is detected on immediately visible surfaces, i.e. a single edge, the existing methods replace the path sample contribution with a user-specified line color c, such that $\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})} = c$. Similarly, when a feature line is detected in a specular reflection, i.e. specular prefixes, the user-specified line color c is "tinted" by the color of the specular surface, such that $\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})} = \prod_{i=1}^{k-1} \alpha_i c$.

In both cases we see that setting,

$$L_e(\mathbf{x}_{k-1}, \mathbf{x}_k) = c, \tag{7}$$

in Eq. (6) preserves the same look-and-feel of existing methods in the context of paths.

In other words, existing methods effectively treat feature lines as a *light source.*

Observation 2: The feature line intersection test can be performed for an arbitrary edge and is not limited to screen-space.

While feature lines are view-dependent by construction, they are not limited to being viewed from the camera. The feature line intersection test of existing methods is performed in the context of a ray-segment. For immediately visible surfaces this is the ray segment from the camera to the first intersection point. For specular prefixes,

[1] A path with specular interactions at every vertex from the sensor up to a given vertex.

these are the ray segments that start at each specular surface and end at each next intersection point.

As path edges are ray-segments, we can use the same algorithm to test for feature line intersection at an arbitrary path edge.

Proposed method. Given these observations, we generalize feature line rendering to paths by,

(1) testing each edge of a path, starting from the sensor, for intersection with feature lines, and
(2) modeling intersected feature lines as *ideal emitters* that absorb all incoming light and emit a user specified color[2].

This process results in *ordinary paths* that preserve the look-and-feel of existing methods while also supporting new physical effects. Furthermore, since they are ordinary paths, we can use them as path samples in an ordinary Monte Carlo estimator Eq. (5).

Algorithm 1: Physically-based feature line rendering

```
Input: Number of path samples per pixel per pass n
/* Without Path Caching                                          */
for pixel ∈ image do
    for i ← 1 to n do
        path = tracePath(pixel)
        path′ = modifyPath(path)                  // Algorithm 2
        image[pixel].accumulate(f(path′)/p(path′))
/* With Path Caching                                             */
for pixel ∈ image do
    for i ← 1 to n do
        path = tracePath(pixel)
        cache[pixel].add(path)
for pixel ∈ image do
    for path ∈ cache[pixel] do
        path′ = modifyPath(path, cache)           // Algorithm 2
        image[pixel].accumulate(f(path′)/p(path′))
```

3.1 Implementation

In this section we provide implementation details and discuss practical considerations of the proposed method. In Algorithms 1 to 3 we provide steps for incorporating feature lines into a path tracer.

3.1.1 Sampling feature line-aware paths. In practice, there are two ways to include feature lines into path sampling: test for feature lines as we generate the path, or generate the path using an out-of-the-box sampler, and then correct it (see Algorithm 2). The former requires implementing a feature line specific path sampler. As feature lines are tested for during path generation, it can be difficult to leverage performance optimizations that cache and re-use other nearby paths. The latter, on the other hand, allows us to use existing path samplers without modification. These paths can be cached [Ogaki and Georgiev 2018] and efficiently re-used to test for feature lines, amortizing the sampling cost, with the downside of potentially generating longer paths than necessary (see Fig. 6).

[2]Black feature lines, as they neither emit nor reflect light, are more precisely described as *ideal absorbers*.

Algorithm 2: Modifying a path to intersect feature lines

```
Function modifyPath(path, cache):
    for edge ∈ path do                    // starting from sensor
        line = intersectLine(edge, cache)      // Algorithm 3
        if line then
            edge.end = emissionVert(line.pos, line.color)
            path = removeEdgesAfter(path, edge)
            break
    return path
```

3.1.2 Computing the PDF of a modified path. When a feature line is found, we modify the structure of the path to end at an emission vertex that represents that feature line. Since we have modified the path, we will need to recompute the probability density.

For path tracing [Kajiya 1986], this is straightforward. The updated PDF can be computed by treating the path as if it was explicitly sampled vertex-by-vertex starting from the sensor. For a modified path $\overline{\mathbf{x}}'$ of length k', its PDF is,

$$p(\overline{\mathbf{x}}') = p(\mathbf{x}_0, \cdots, \mathbf{x}_{k'}) = p(\mathbf{x}_0)p(\mathbf{x}_1|\mathbf{x}_0)\prod_{i=2}^{k'} p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{x}_{i-2})\,. \quad (8)$$

Note that the PDF does not take into account the stochastic sampling of rays used to test for the feature line. This is essentially a stochastic intersection test, and, if none of the ray samples find the closest feature line, the updated path will be biased. We further quantify and discuss this bias in Section 3.1.5.

Other path sampling methods require more care. For example, in light tracing, paths are sampled vertex-by-vertex starting from a light source. As an out-of-the-box light tracing path sampler is unaware of feature lines as light sources, it will never generate a path starting at a feature line. Solving for the probability density of a modified path $\overline{\mathbf{x}}'$ would require marginalization that often has no closed-form solution in practice.

We leave the exploration of other path sampling methods to future work.

3.1.3 Feature line width. We can express the width of feature lines in several ways, and each will affect how we test for feature line intersections down a path. Feature line width expressed in object-space will maintain a consistent width in 3D space, and the closer they are to the camera the larger they will appear. Feature line width expressed in screen-space, on the other hand, maintains a consistent width on the screen regardless of how far objects are from the camera.

To view screen-space lines from the camera, we can project the screen-space line width into 3D space, forming a cone [Choudhury and Parker 2009]. This works well for immediately visible surfaces, however, preserving screen-space width for reflections and edges deeper in a path is less straightforward. For example, if we are viewing feature lines reflected by a sphere, we would need to account for the surface curvature of the encapsulated region (see Fig. 7). In general, it can be challenging to calculate the exact region of surface to test against, and the problem compounds at edges deeper in a path.

Algorithm 3: Feature line intersection test[3]

Input: m number of samples for the intersection test
Output: Line struct with position, depth, and color of the intersected feature line, or **null** if none found

Function `intersectLine` *(edge, cache)*:
  $region$ = `computeTestRegion`($edge$) // Section 3.1.3
  $samples$ = `sampleRegion`($region, cache, m$)
  $closest$ = `Line`(**null**, $+\infty$, **null**)
  for $\mathbf{s} \in samples$ **do**
    $line$ = `lineMetric`($edge, \mathbf{s}$)
    if $line$ **and** $line.depth < closest.depth$ **then**
      $closest = line$
  return closest
Function `lineMetric` *(edge, s)*:
  $\mathbf{p} = edge.start$, $\mathbf{q} = edge.end$
  if `satisfiesMetric`($\mathbf{q}, \mathbf{s}$) **then** // Section 3.1.4
    $\mathbf{x} = \underset{\mathbf{v}\in\{\mathbf{q},\mathbf{s}\}}{\text{argmin}} \|\mathbf{p} - \text{projectOnto}(\mathbf{v}, edge)\|$
    $\mathbf{q}' = \text{projectOnto}(\mathbf{x}, edge)$
    return `Line`($\mathbf{q}'$, $\|\mathbf{p} - \mathbf{q}'\|$, $\mathbf{x}.lineColor$)
  return null

The method of Ogaki and Georgiev [2018] overcomes this issue for paths with specular prefixes by comparing projected path samples directly in screen-space (see Fig. 7). Unfortunately, this method does not work for paths with diffuse and glossy reflections.

Rather than try to compute an exact solution for preserving screen-space line width for arbitrary edges, we propose a heuristic that works reasonably well in practice. The 2D screen-space line widths can be projected into 3D object-space forming a cone. This cone can be extended down the path to approximately preserve the screen-space line width coherency,

$$w_{scaled} = d \cdot p_{width} \cdot w_{screen}\,, \tag{9}$$

where w_{scaled} is the line width scaled to object-space, d is the distance down the path edges from the first vertex to the point on the path where we want to measure the line width, p_{width} is the pixel width in object-space at a distance of 1 from the camera, and w_{screen} is the line width in screen-space.

Given this heuristic the feature line test region of Algorithm 3 is a cone-segment, where the opening and closing radii are the scaled line widths at the starting and ending vertices of the edge respectively. When using variable line width the radii of the cone segment should be large enough to encapsulate the largest possible line width.

Some care needs to be taken to make sure ray samples generated with a cone-segment do not start below the surface. In our implementation, a ray is sampled by first sampling a point on the opening disk and then computing the direction towards the corresponding point on the closing disk. Once a ray sample is generated we move its origin forward down the ray until it is no longer below the surface. Ray samples that both start below and face away from the surface are rejected.

[3]For brevity we consider a single line width. Testing for varying line widths requires verifying independently if the lines at $\mathbf{q}$ and $\mathbf{s}$ would intersect the edge.

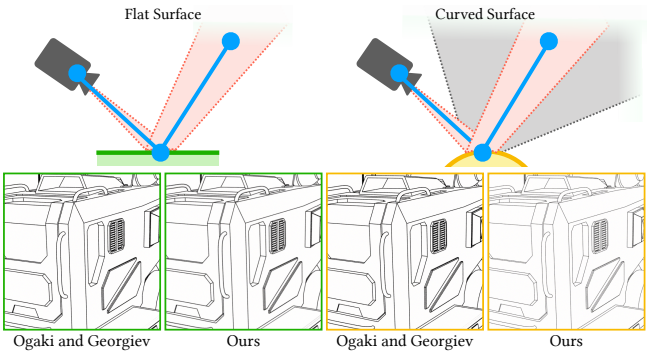

Fig. 7. The heuristic presented in Eq. (9) can potentially over or underestimate the screen-space line width. For objects reflected by a flat surface both the method of Ogaki and Georgiev [2018] and our heuristic preserve screen-space line width. For curved surfaces Ogaki and Georgiev [2018]'s method (gray) correctly accounts for surface curvature while our heuristic (red) underestimates the line width.

3.1.4 Feature line metrics. Different metrics can be used to visualize different discontinuities. If a single metric is unable to achieve the desired style it is possible to combine several into a composite metric (see Fig. 2). In Fig. 8 we demonstrate the metrics used for the results in this paper over a range of parameters.

For the results in this paper we use a variation of the metrics proposed by Choudhury and Parker [2009]; Ogaki and Georgiev [2018] and combine them by taking their max term (i.e. if any metric would return true the combined metric returns true):

$$\text{MeshID}: \quad id_{\mathbf{q}} \neq id_{\mathbf{s}} \tag{10}$$

$$\text{Albedo}: \quad |a_{\mathbf{q}} - a_{\mathbf{s}}| > t_{albedo} \tag{11}$$

$$\text{Normal}: \quad (1.0 - \vec{n}_{\mathbf{q}} \cdot \vec{n}_{\mathbf{s}}) > t_{normal} \tag{12}$$

$$\text{Depth}: \quad |d_{\mathbf{q}} - d_{\mathbf{s}}| > t_{depth} \tag{13}$$

where we compare an edge represented by a start vertex $\mathbf{p}$, an end vertex $\mathbf{q}$, and a ray $\vec{\mathbf{pq}}$, to a sample vertex $\mathbf{s}$, the variables $id_{\square}$, $a_{\square}$, $\vec{n}_{\square}$, and $d_{\square}$ are the mesh ID, albedo, surface normal, and depth for a given vertex, and $t_{\square}$ are user-defined, metric-specific thresholds.

Choudhury and Parker [2009] suggest that the depth threshold t_{depth} should be implemented as a function. For the results in this paper we use the following heuristic with $\beta = 2$,

$$t_{depth} = \beta \min(d_{\mathbf{q}}, d_{\mathbf{s}}) \|\vec{\mathbf{ps}} - \vec{\mathbf{pq}}\| / |\vec{\mathbf{pq}} \cdot \vec{n}_{closest}|. \tag{14}$$

where β is a user-defined scaling factor, and $\vec{n}_{closest}$ is the surface normal of either $\mathbf{q}$ or $\mathbf{s}$, whichever is closer to the edge start vertex $\mathbf{p}$. This heuristic scales with both the distance from the origin of the query ray, as well as the angle incident to the surface at the intersection point, reducing overly dense lines on far away, finely detailed geometry and on surfaces near parallel to the query ray.

3.1.5 Approximating the feature line test. A ray can potentially intersect zero or more feature lines. To accurately determine the closest feature line a ray intersects, we must compare the ray's hit

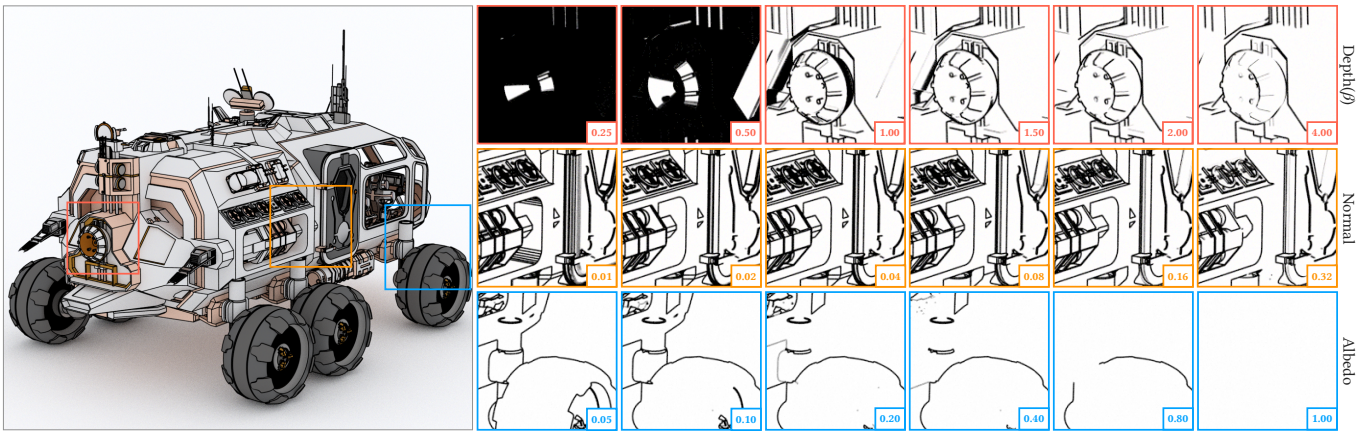

Fig. 8. Artists can control the level of detail of rendered lines by adjusting the threshold of the line metrics. Here we demonstrate a range of common thresholds for the depth, normal, and albedo metrics in Eqs. (11) to (13). We have found depth $\beta = 2.0$, $t_{normal} = 0.08$, and $t_{albedo} = 0.1$ work well in practice.

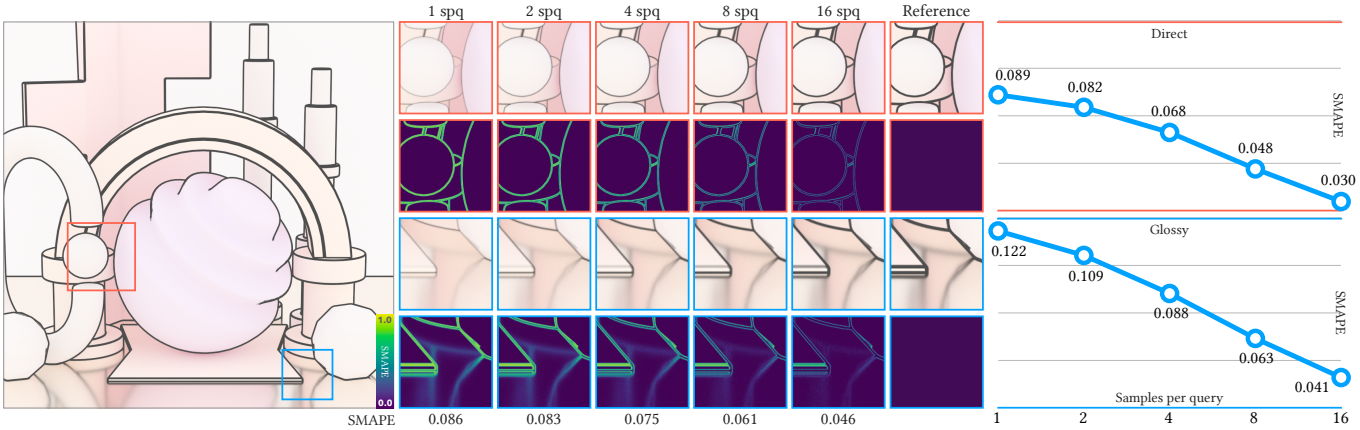

Fig. 9. Using a finite number of ray samples to estimate the feature line test results in visible bias that manifests as feature lines appearing thinner and partially transparent. Increasing the number of samples reduces this bias, but at the cost of more computational overhead. In practice we found that 16-32 ray samples works well for most rendering scenarios, with thicker lines tending to require more samples than thinner lines.

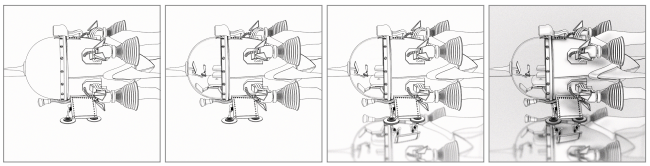

Fig. 10. Testing for feature lines at every edge of a path can result in a shading-like effect that is undesirable in some applications. This can be alleviated by limiting which surfaces and materials reflect feature lines. Image (a) tests for feature lines on only immediately visible surfaces, (b) adds specular reflection and refraction, (c) adds glossy reflections, and (d) reflects feature lines on all surfaces. In particular, image (d) appears more like a shaded image than a line drawing.

point to all of the points that lie within some portion of surface (see Fig. 3b). Comparing to all points for an exact solution is not possible in practice.

Existing ray-based feature line methods, as well as ours, approximate this with a finite number of samples. Given some conditions hold (see Appendix A), as we keep increasing the number of samples we will eventually find the closest feature line. The more samples we use, the more likely we are to have sampled a location that results in the correct feature line, and as a result, the more accurate the render is.

However, this finite sample approximation introduces bias. In practice, the bias introduced by this approximation is manifested as lines being slightly transparent, allowing lines below them, or the surface below them, to be partially visible. In Fig. 9 we quantify this bias over several difference sample counts and empirically demonstrate that the bias diminishes as the sample count is increased.

3.1.6 Selective reflection of feature lines. By treating feature lines as light sources, some areas of the rendered image, particularly diffuse surfaces, may end up with a shaded appearance according to the feature line's color. In some artistic applications, this can be undesirable. One way to overcome this is to specify which surfaces and materials reflect feature lines. In our implementation we specify this as a boolean shader flag, and stop feature line detection for a path at the first vertex that does not reflect feature lines. In Fig. 10 we demonstrate the effect of limiting feature line reflection.

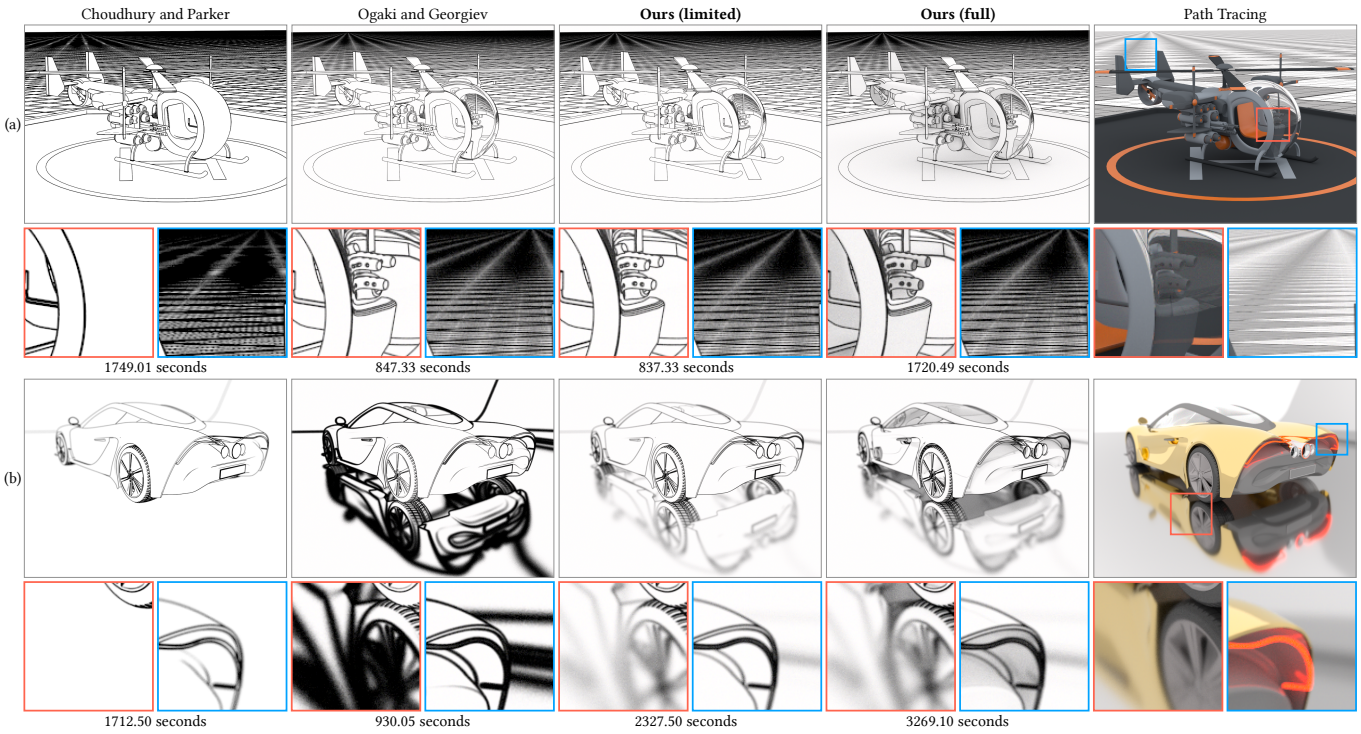

Fig. 11. We compare two variants of our method, one that limits feature line tests to certain path edges (Ours (limited)) , and one that tests all path edges (Ours (full)), to the methods of Choudhury and Parker [2009] and Ogaki and Georgiev [2018]. The method of Choudhury and Parker [2009] lacks specular and glossy reflections, but handles depth-of-field reasonably well. The method of Ogaki and Georgiev [2018] is significantly more performant due to efficient sample re-use, but has strong artifacts in areas that are out of focus, or have glossy reflections. Our method handles both robustly. Rightmost images show the path traced result.

4 RESULTS AND DISCUSSION

In this section we compare our method against existing work in several scenes representative of common rendering scenarios. We additionally demonstrate the robustness of our method under varying intensities and combinations of different physical effects.

We implemented our method on a CPU-based spectral renderer, using unidirectional path tracing [Kajiya 1986] for the path sampler, MIS spectral importance sampling for wavelength sampling [Wilkie et al. 2014; West et al. 2020], and RGB assets were spectral upsampled [Jakob and Hanika 2019]. All images in this paper were rendered on a single workstation with a 10-core Intel i9-7900X and 32GB of 2666 Mhz DDR4 memory. The implementation of our method supports variable line width and color, programmable line metrics, and the screen-space cache of Ogaki and Georgiev [2018].

Comparison to existing methods. In Fig. 11 we compare our method to the work of Choudhury and Parker [2009] and Ogaki and Georgiev [2018] on two different scenes. In row (a) we provide a demonstration of the capabilities of each method on a simple scene with only specular reflection and refraction. In row (b) we provide a challenging scenario for existing work with strong physical effects that demonstrate the capabilities of our method.

In Fig. 11 (a), for Ours (limited), feature lines are rendered only for those paths supported by Ogaki and Georgiev [2018]. For Ours (full), feature lines are tested at every edge of every path. Each render is 960 by 960 pixels, received 500 path samples per pixel, and approximately 10 ray samples per feature line test. The path traced image provides a representation of the scene's lighting, geometry, and materials.

Here, the method of Choudhury and Parker [2009] renders clean lines for immediately visible surfaces, but lacks feature lines seen through specular and glossy reflections (see the red inset). Additionally, their method shows aliasing artifacts along the far-plane of the rendered image due to the fixed shape of the ray stencil (see the blue inset). The method of Ogaki and Georgiev [2018] adds feature lines for specular interactions, shows significant performance improvement over the method of Choudhury and Parker [2009], and, due to their stochastic sampling process, does not suffer from aliasing artifacts. Ours (limited) has near-identical performance to Ogaki and Georgiev [2018] since we can selectively leverage the same path caching scheme when appropriate. Ours (full) performs the feature line test for every edge of every path, resulting in a shaded effect, especially on diffuse and rough glossy surfaces.

In Fig. 11 (b) the methods of Choudhury and Parker [2009] and Ogaki and Georgiev [2018] were used for feature line detection on a scene with strong depth-of-field and glossy reflection effects. Specifically for the method of Ogaki and Georgiev [2018], the screen-space cache stored paths with glossy reflection in addition to those with specular reflection and refraction. For Ours (limited), feature lines are rendered for the paths supported by Ogaki and Georgiev [2018] and additionally glossy reflections.

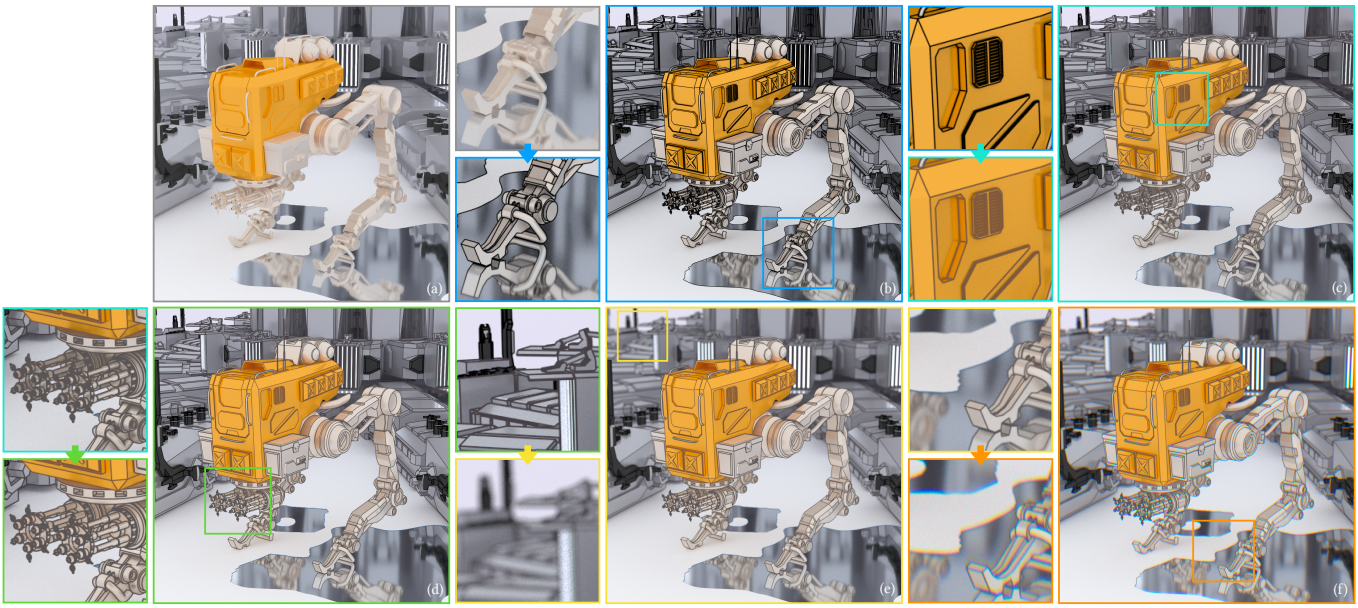

Fig. 12. Different line settings and physical effects can be combined creatively to achieve different artistic expressions. Here, (a) is path traced, (b) adds feature lines, (c) specifies line widths and colors, (d) limits feature line reflection to only some surfaces, (e) introduces a physical lens model, and (f) adds wavelength-dependent refraction to the lens. The insets highlight areas of visible change from each preceding image.

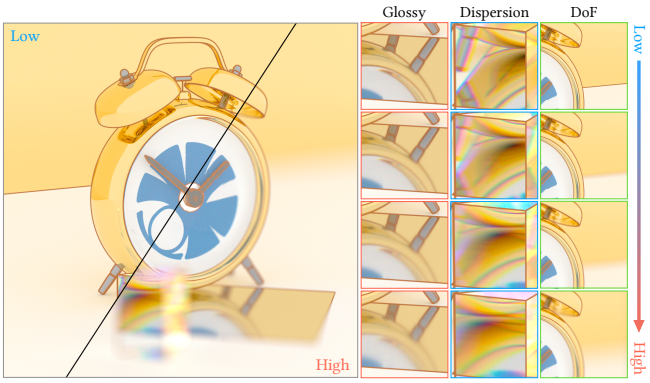

Fig. 13. Our method is capable of rendering feature lines even in the presence of strong physical effects. Here we demonstrate how material roughness, wavelength-dependent scattering, and different lens apertures affect how feature lines are rendered.

The method of Choudhury and Parker [2009], due to explicitly sampling rays for each ray stencil, plausibly renders feature lines in out-of-focus regions, but lacks the glossy reflection on the floor. The method of Ogaki and Georgiev [2018], due to its screen-space formulation, produces strong artifacts for both out-of-focus regions and the glossy reflection on the floor. In contrast, our method successfully captures the physical aspects of the scene in the rendered feature lines.

Artistic options. Fig. 12 demonstrates how various effects can be creatively combined to produce compelling rendered images. Image (a) shows the scene rendered as-is with path tracing. Image (b) uses our method to render feature lines at every bounce of every path with a single line width and color. Image (c) introduces variable line width and color, resulting in some surfaces being visibly affected by the color of the feature lines. To alleviate this, in image (d), we set the material flags to only reflect feature lines on certain surfaces, cleaning up much of the "shading" effect of our method. Image (e) changes from a pinhole camera to a physical lens model, causing the portions of the scene that are far from the focal plane to defocus. Lastly, image (f) adds in wavelength-dependent refraction to the lens, causing a subtle rainbow effect around the edges of objects. The insets of each image highlight areas of significant change from the preceding image.

Intensity of effect. In Fig. 13, we demonstrate how varying intensities of physical effects affect feature lines rendered with our method. In the first column of insets, we adjust the roughness parameter of the plane on the floor to simulate varying levels of glossiness. In the second column, we modulate the wavelength-dependent index of refraction of the prism to simulate materials with varying amounts of angular dispersion. In the third column, we change the lens aperture diameter to defocus parts of the scene further from the focal plane. In particular, the second column of insets show feature lines of the alarm clock seen through wavelength-dependent refraction and glossy reflection. Our path-based method makes rendering feature lines for these types of complex interactions possible.

5 LIMITATIONS AND FUTURE WORK

The method we present in this paper samples paths vertex-by-vertex, starting from the sensor. This makes computing the PDF of modified paths straightforward, but can be an inefficient path sampling method for scenes with complex occlusion and lighting. Other path sampling methods, such as bidirectional path tracing, excel at such scenes, but computing the probability density of modified paths under these methods can be challenging. Exploring the feasibility

of using new path sampling methods in our feature line rendering method would be an interesting avenue of research.

Preserving screen-space line width of feature lines seen through reflections can be challenging due to variations in surface curvature. We present one heuristic that extends the feature line width down the entire path. This heuristic works well in practice, but can result in some feature lines, particularly those at deeper edges of a path, being drawn thicker or thinner than desired. Feature line tests that more accurately approximate screen-space line width at deeper edges of paths would improve line width coherency.

Ray casting for feature line tests can be expensive. Ogaki and Georgiev [2018] introduce screen-space caching that works well for immediately visible surfaces and specular reflection, but provides little benefit for feature line tests at edges deeper in a path. One potential solution is to cache path vertices in a spatial acceleration structure, such as a Bounding Volume Hierarchy (BVH), and use a cone-segment to query for cached vertices [Wiche and Kuri 2020]. Such cone-BVH queries, however, are generally more computationally expensive than ray-BVH queries. Exploration of the trade-off between ray-BVH queries and cone-BVH queries in the context of feature line rendering could lead to improved performance.

As discussed by [Ogaki and Georgiev 2018], correct rendering of feature lines for occluded surfaces is challenging. These surfaces are not visible from the query ray during a feature line intersection test, and, though their feature lines may intersect the query ray, will not be included in the region tested. One solution is to continue each query ray through each surface, gathering all intersection points along the ray, but this can significantly increase computational cost. New methods to efficiently determine feature line intersections for occluded surfaces would improve the robustness and temporal coherency of feature line rendering.

Modeling feature lines as light sources preserves the look-and-feel of the existing methods, and opens up feature lines to new physical effects. However, this model limits the style of feature lines that can be rendered. In some artistic applications it may be desirable to render feature lines that have transparency or even reflect light. A formulation that supports such additional functionality would further improve the creative expression of physically-based feature lines.

Cole et al. [2006] explore *stylized focus*, a method for directing the viewer's gaze in stylized renderings. Some of the effects presented in this paper, such as lens blur and spectral dispersion, de-emphasize regions of the rendered image and can potentially be used in applications of stylized focus. Further study of the perceptual aspects of the proposed method would assist artists in evaluating if it meets their needs.

6 CONCLUSION

In this paper we presented a novel, path-based method that brings feature line rendering, a classically non-photorealistic technique, into PBR by modeling feature lines as view-dependent, implicit light sources. This expands feature line rendering to a wide variety of physical effects, such as lens blur and dispersion, that were previously approximated through screen-space post-processes and compositing. Those effects can now be accurately simulated and seamlessly combined, opening up new avenues of creative expression for artists. We believe that the method presented in this paper is a step towards unifying the expressiveness of NPR and the correctness of PBR, and that there is great potential for future research in this direction.

ACKNOWLEDGMENTS

We express our gratitude to the following people: the reviewers, Nobuyuki Umetani, Sayan Mukherjee, Daniel Meister, Sabyasachi Mukherjee, Jamorn Sriwasansak, and Aaryaman Vasishta for their valuable feedback and suggestions, and Sayan Mukherjee for their assistance with the proof in Appendix A.

We also express our gratitude to the following Blend Swap users: *Wundersound* for the battle sprinter in Figs. 1, 7 and 12, *DimitrisC* for the UFO in Figs. 1 and 12, *ruwo* for the glass jars in Fig. 3, *Wig42* for the breakfast room in Fig. 3, *vajrablue* for the sci-fi rover in Fig. 8, *PIXX0* for the abstract shapes in Fig. 9, *thecali* for the spaceship in Fig. 10, *timsh* for the helicopter in Fig. 11a, *Jorgeferrer* for the car in Fig. 11b, and *chams* for the alarm clock in Fig. 13.

This work has been partially funded by a grant from Autodesk and a Japanese Government (Monbukagakusho - MEXT) Scholarship.

REFERENCES

Andreas Bauer. 2017. A New Contour Method for Highly Detailed Geometry. In *ACM SIGGRAPH 2017 Talks* (Los Angeles, California) *(SIGGRAPH '17)*. Association for Computing Machinery, New York, NY, USA, Article 71, 2 pages. https://doi.org/10.1145/3084363.3085052

Dennis R. Bukenberger, Katharina Schwarz, and Hendrik P. A. Lensch. 2018. Stereo-Consistent Contours in Object Space. *Computer Graphics Forum* 37, 1 (2018), 301–312. https://doi.org/10.1111/cgf.13291 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13291

Pierre Bénard and Aaron Hertzmann. 2019. Line Drawings from 3D Models: A Tutorial. *Foundations and Trends® in Computer Graphics and Vision* 11, 1-2 (2019), 1–159. https://doi.org/10.1561/0600000075

A. N. M. Imroz Choudhury and Steven G. Parker. 2009. Ray Tracing NPR-Style Feature Lines. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (New Orleans, Louisiana) *(NPAR '09)*. Association for Computing Machinery, New York, NY, USA, 5–14. https://doi.org/10.1145/1572614.1572616

Forrester Cole, Douglas DeCarlo, Adam Finkelstein, Kenrick Kin, R. Morley, and Anthony Santella. 2006. Directing Gaze in 3D Models with Stylized Focus. 377–387. https://doi.org/10.2312/EGWR/EGSR06/377-387

Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. 2008. Where Do People Draw Lines? *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–11. https://doi.org/10.1145/1360612.1360687

Doug DeCarlo. 2012. Depicting 3D shape using lines. In *Human Vision and Electronic Imaging XVII*, Bernice E. Rogowitz, Thrasyvoulos N. Pappas, and Huib de Ridder (Eds.), Vol. 8291. International Society for Optics and Photonics, SPIE, 361 – 376. https://doi.org/10.1117/12.916463

Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. 2004. Interactive Rendering of Suggestive Contours with Temporal Coherence. In *Third International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. 15–24.

Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. 2003. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 22, 3 (July 2003), 848–855.

Aaron Hertzmann. 2020. Why Do Line Drawings Work? A Realism Hypothesis. *Perception* 49, 4 (2020), 439–451. https://doi.org/10.1177/0301006620908207 arXiv:https://doi.org/10.1177/0301006620908207 PMID: 32126897.

Aaron Hertzmann. 2021. The Role of Edges in Line Drawing Perception. *Perception* 50, 3 (2021), 266–275. https://doi.org/10.1177/0301006621994407 arXiv:https://doi.org/10.1177/0301006621994407 PMID: 33706622.

Wenzel Jakob and Johannes Hanika. 2019. A Low-Dimensional Function Space for Efficient Spectral Upsampling. *Computer Graphics Forum (Proceedings of Eurographics)* 38, 2 (March 2019).

Tilke Judd, Frédo Durand, and Edward H. Adelson. 2007. Apparent ridges for line drawing. *ACM Trans. Graph.* 26, 3 (2007), 19.

James T. Kajiya. 1986. The Rendering Equation. 20, 4 (Aug. 1986), 143–150. https://doi.org/10/cvf53j

Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. 2008. Line-Art Illustration of Dynamic and Specular Surfaces. *ACM Trans. Graph.* 27, 5, Article 156 (Dec. 2008), 10 pages. https://doi.org/10.1145/1409060.1409109

Yunjin Lee, Lee Markosian, Seungyong Lee, and John F. Hughes. 2007. Line Drawings via Abstracted Shading. In *ACM SIGGRAPH 2007 Papers* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 18–es. https://doi.org/10.1145/1275808.1276400

Difan Liu, Mohamed Nabail, Aaron Hertzmann, and Evangelos Kalogerakis. 2020. Neural Contours: Learning to Draw Lines From 3D Shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shinji Ogaki and Iliyan Georgiev. 2018. Production Ray Tracing of Feature Lines. In *SIGGRAPH Asia 2018 Technical Briefs* (Tokyo, Japan) *(SA '18)*. Association for Computing Machinery, New York, NY, USA, Article 15, 4 pages. https://doi.org/10.1145/3283254.3283273

Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 2004. Ridge-Valley Lines on Meshes via Implicit Surface Fitting. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 609–612. https://doi.org/10.1145/1015706.1015768

Matt Pharr, Brent Burley, Per Christensen, Marcos Fajardo, Luca Fascione, and Christopher Kulla. 2018. Design and Implementation of Modern Production Renderers. In *ACM SIGGRAPH 2018 Panels* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.1145/3209621.3214901

Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation* (3 ed.). San Francisco, CA, USA.

Takafumi Saito and Tokiichiro Takahashi. 1990. Comprehensible Rendering of 3-D Shapes. *ACM Siggraph Computer Graphics* 24, 197–206. https://doi.org/10.1145/97879.97901

Eric Veach. 1997. *Robust Monte Carlo Methods for Light Transport Simulation.* Ph.D. Thesis. Stanford University, United States – California.

Rex West, Iliyan Georgiev, Adrien Gruson, and Toshiya Hachisuka. 2020. Continuous Multiple Importance Sampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39, 4 (July 2020). https://doi.org/10.1145/3386569.3392436

Roman Wiche and David Kuri. 2020. Performance Evaluation of Acceleration Structures for Cone-tracing Traversal. *Journal of Computer Graphics Techniques Vol* 9, 1 (2020).

Alexander Wilkie, Sehera Nawaz, Marc Droske, Andrea Weidlich, and Johannes Hanika. 2014. Hero Wavelength Spectral Sampling. 33, 4 (June 2014), 123–131. https://doi.org/10/f6fgb4

A STOCHASTIC APPROXIMATION OF FEATURE LINES

The stochastic approximation of the feature line intersection test results in bias. Let us show that this bias vanishes as the number of samples k grows large.

Let $m : \mathbb{R}^n \to \mathbb{R}$ denote the Lebesgue measure. Let $\mathbf{q}, \mathbf{r} \in \mathbb{R}^3$ denote vertices, and let $\vec{\mathbf{qr}}$ denote the ray from $\mathbf{q}$ to $\mathbf{r}$. Let S be the finite area region of surface encapsulated by the feature line intersection test for some ray $\vec{\mathbf{qr}}$, and T be the sub-region that, when tested against produces the unoccluded feature line closest to $\mathbf{q}$, such that $T \subset S$ and $0 < m(T) < m(S) < \infty$. Let $f : S \to \mathbb{R}$ be a function constant on T that maps vertices $\mathbf{x}$ to a feature line identifier c. Let us assume there is a feature line identifier $c \in \mathbb{R}$ such that $f(\mathbf{x}) = c$ for all $\mathbf{x} \in T$. Let $p : S \to \mathbb{R}$ be a probability density function such that $p(\mathbf{x}) > 0$ for all $\mathbf{x} \in S$, and let $\epsilon = P(\mathbf{x} \in T) = \int_T p(\mathbf{x})\mathrm{d}\mathbf{x}$.

If we independently select k vertices $\mathbf{x}_1, \cdots, \mathbf{x}_k \sim p(\mathbf{x})$ from S,

$$P(c \notin \{f(\mathbf{x}_1), \cdots, f(\mathbf{x}_k)\}) = \prod_{i=1}^{k} P(f(\mathbf{x}_i) \neq c) \tag{15}$$

$$\leq (1-\epsilon)^k \tag{16}$$

$$\leq \exp(-\epsilon k)\,, \tag{17}$$

since for any i, $P(f(\mathbf{x}_i) = c) \geq P(\mathbf{x}_i \in T) = \epsilon$.

Therefore, we can say that $c \in \{f(\mathbf{x}_1), \cdots, f(\mathbf{x}_k)\}$ happens with high constant probability when $k \sim O(1/\epsilon)$, and, as both S and T have finite, non-zero measure,

$$\lim_{k\to\infty} P(c \notin \{f(\mathbf{x}_1), \cdots, f(\mathbf{x}_k)\}) = 0\,, \tag{18}$$

and the bias vanishes as k grows large.

B MODELING FEATURE LINES AS LIGHT SOURCES

We show that, for a limited subset of paths, the setting the emission term $L_e(\mathbf{x}_{k-1}, \mathbf{x}_k)$ to the user-specified line color c preserves the behavior of existing methods.

We assume paths are sampled vertex-by-vertex starting from the sensor, with perfect importance sampling for the sensor responsivity term W, the geometry terms G, and BSDF terms ρ_s. Furthermore, we assume a perspective camera, though the following holds for other camera models where perfect importance sampling is possible.

B.1 Immediately visible surfaces

$$\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})} = \frac{\rho(\mathbf{x}_0)\eta(\mathbf{x}_0,\mathbf{x}_1)\rho(\mathbf{x}_1)}{p(\mathbf{x}_0,\mathbf{x}_1)} \tag{19}$$

$$= \frac{W(\mathbf{x}_0,\mathbf{x}_1)G(\mathbf{x}_0,\mathbf{x}_1)L_e(\mathbf{x}_0,\mathbf{x}_1)}{p_W(\mathbf{x}_0,\mathbf{x}_1)\frac{D(\mathbf{x}_1,\mathbf{x}_0)}{||\mathbf{x}_0-\mathbf{x}_1||^2}} \tag{20}$$

$$= \frac{\frac{1}{A_i A_l D(\mathbf{x}_0,\mathbf{x}_1)^4}\frac{D(\mathbf{x}_0,\mathbf{x}_1)D(\mathbf{x}_1,\mathbf{x}_0)}{||\mathbf{x}_0-\mathbf{x}_1||^2}L_e(\mathbf{x}_0,\mathbf{x}_1)}{\frac{1}{A_i A_l D(\mathbf{x}_0,\mathbf{x}_1)^3}\frac{D(\mathbf{x}_1,\mathbf{x}_0)}{||\mathbf{x}_0-\mathbf{x}_1||^2}} \tag{21}$$

$$= L_e(\mathbf{x}_0,\mathbf{x}_1)\,, \tag{22}$$

where $\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})}$ is the path sample contribution, A_i is the surface area of the pixel, A_l is the surface area of the lens, $p_W(\mathbf{x},\mathbf{y})$ is the sensor responsivity pdf for the the vertex $\mathbf{x}$ and the direction $\omega_{\mathbf{xy}}$. For other terms, see Section 2.3.

In the case of immediately visible surfaces, by setting $L_e(\mathbf{x}_0,\mathbf{x}_1) = c$ the path sample contribution will be exactly the line color c which preserves the behavior of both [Choudhury and Parker 2009] and [Ogaki and Georgiev 2018] as desired.

B.2 Specular prefixes

$$\frac{f(\overline{\mathbf{x}})}{p(\overline{\mathbf{x}})} = \frac{\prod_{i=1}^{k}\eta(\mathbf{x}_{i-1},\mathbf{x}_i)\prod_{i=0}^{k}\rho(\mathbf{x}_i)}{p(\mathbf{x}_0,\cdots,\mathbf{x}_k)} \tag{23}$$

$$= \frac{W(\mathbf{x}_0,\mathbf{x}_1)\prod_{i=1}^{k}G(\mathbf{x}_{i-1},\mathbf{x}_i)\prod_{i=1}^{k-1}\rho_s(\mathbf{x}_i)L_e(\mathbf{x}_{k-1},\mathbf{x}_k)}{p_W(\mathbf{x}_0,\mathbf{x}_1)\frac{D(\mathbf{x}_1,\mathbf{x}_0)}{||\mathbf{x}_0-\mathbf{x}_1||^2}\prod_{i=2}^{k}p(\mathbf{x}_i|\mathbf{x}_{i-1},\mathbf{x}_{i-2})} \tag{24}$$

$$= \frac{\frac{1}{A_i A_l D(\mathbf{x}_0,\mathbf{x}_1)^4}D(\mathbf{x}_0,\mathbf{x}_1)\prod_{i=1}^{k}\frac{D(\mathbf{x}_i,\mathbf{x}_{i-1})}{||\mathbf{x}_{i-1}-\mathbf{x}_i||^2}}{\frac{1}{A_i A_l D(\mathbf{x}_0,\mathbf{x}_1)^3}\prod_{i=1}^{k}\frac{D(\mathbf{x}_i,\mathbf{x}_{i-1})}{||\mathbf{x}_{i-1}-\mathbf{x}_i||^2}}$$
$$\frac{\prod_{i=1}^{k-1}\alpha_i\delta(1-\omega_i\cdot r(\omega_{i-1}))L_e(\mathbf{x}_{k-1},\mathbf{x}_k)}{\prod_{i=1}^{k-1}\delta(1-\omega_i\cdot r(\omega_{i-1}))} \tag{25}$$

$$= \prod_{i=1}^{k-1}\alpha_i L_e(\mathbf{x}_{k-1},\mathbf{x}_k) \tag{26}$$

where ω_i is the direction from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$, $r(\omega)$ is the appropriate reflected or refracted direction for ω. Note the geometry term G does not have a D term for specular interactions.

In the case of specular prefixes, by setting $L_e(\mathbf{x}_{k-1},\mathbf{x}_k) = c$, and only requiring that $0 \leq \alpha_i \leq 1$, Eq. (26) preserves the "color tinting" behavior of Ogaki and Georgiev [2018]. Additionally, when α_i are all 1, Eq. (26) simplifies to exact color replacement.