CHAPTER 10

REFRACTION RAY CONES FOR TEXTURE LEVEL OF DETAIL

Jakub Boksansky, Cyril Crassin, and Tomas Akenine-Möller
NVIDIA

ABSTRACT

Texture filtering is an important implementation detail of every rendering system. Its purpose is to achieve high-quality rendering of textured surfaces, while avoiding artifacts, such as aliasing, Moiré patterns, and unnecessary overblur. In this chapter, we extend the ray cone method for texture level of detail so that it also can handle refraction. Our method is suitable for current game engines and further bridges the gap between offline rendering and real-time ray tracing.

10.1 INTRODUCTION

Accurate rendering of transparent and semitransparent surfaces is one of the many appealing features of renderers based on ray tracing. By simply following the paths of rays refracted on transmissive surfaces, it is possible to render materials, such as glass and water, with great realism. Ray tracing enables us to take refraction indices of materials on both sides of the surface into account and "bend" the rays in a physically correct way, which is an effect that is difficult to achieve using rasterization and which contributes significantly to the realistic rendering of semitransparent materials. Here, we focus on using the ray cone method [1, 2] for the filtering of textures on surfaces seen by refracted rays to achieve the same level of visual quality as for reflected and primary rays. An example is shown in Figure 10-1.

The geometry of perfect refraction (and reflection) has been covered in detail in Chapters 8 and 9 and by de Greve [5], including the interesting case of total internal reflection and the relation of Fresnel equations to the modeling of transmissive surfaces. A practical implementation in a ray tracer is covered in Chapter 11.

A. Marrs, P. Shirley, I. Wald (eds.), *Ray Tracing Gems II*, https://doi.org/10.1007/978-1-4842-7185-8_10

Figure 10-1. *Refractive materials, such glass and water, can be rendered accurately using ray tracing with respect to given indices of refraction. This is a significant improvement compared to rasterization. This picture was rendered using a fast path tracer, which correctly integrates over all necessary domains in order to reproduce refraction effects as well as camera defocus blur.*

A more complex case is the modeling of refractions on rough surfaces, e.g., frosted glass and ice, where rays are not refracted perfectly, but their direction is randomized. A model for such materials is described by a bidirectional transmittance distribution function (BTDF). A formal formulation of such a BTDF employing a widely used Cook–Torrance microfacet model was provided in the seminal paper by Stam [9], and importance sampling was added by Walter et al. [10]. A more efficient importance sampling method was later developed by Heitz [7].

Though the problem of ray cone refraction might at first seem to be the same as reflection, the fundamental difference lies in the fact that indices of refraction are typically different for bodies at opposing sides of the hit point surface, whereas they are always the same for reflecting ray cones. This introduces the relative index of refraction η, which has significant impact on the refracted ray direction. It is not only the width of the cone that changes, but also its geometry. Depending on the relative index of refraction (whether a ray refracts from an optically denser to a thinner environment or vice versa), the cones can either shrink or grow. Also, the centerline of the refracted cone can differ significantly from the direction of the refracted ray. Note that altering the refracted ray direction to reflect this change is not viable, in our

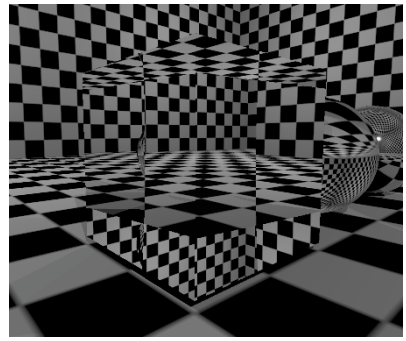

Figure 10-2. *A glass cube showing refraction in the center and TIR closer to the sides.*

opinion, as it could miss geometry that would be hit under certain angles, essentially sampling a correct footprint from incorrect geometry. Such a solution would be more expensive as well.

When the ray cone refracts from an optically denser to a thinner environment, total internal reflection (TIR) can occur. A good real life example is *Snell's window* visible when an observer underwater looks up to the surface. Another example is shown on the glass cube in Figure 10-2. For incident rays making an angle with a normal that is larger than a certain critical angle, a perfect reflection occurs and we can use the ray cone solution for reflected rays. In those cases, no refraction occurs. A problem occurs on the boundary between a fully refracted and a totally internally reflected (TIRed) ray, when the ray is incident under an angle close to the critical angle. In this case, one part of the ray cone is reflected and other part is TIRed (see Figure 10-3).

The method presented in this chapter calculates an approximate refracted ray cone that attempts to match the footprint of a "perfectly refracted" cone as closely as possible without altering the refracted ray direction. As in other methods [3, 6, 8], we also only follow the part of the cone where the central ray ends up; i.e., if it is refracted, we construct a refracted ray cone, and if the central ray is TIRed, we construct a reflected ray cone.

10.2 OUR METHOD

A ray cone is defined by a width w, a spread angle α, a ray origin O, and a direction $\mathbf{d}$ [2], as illustrated in Figure 10-4, left. The curvature approximation at the hit point is modeled as a signed angle β, where a positive sign indicates

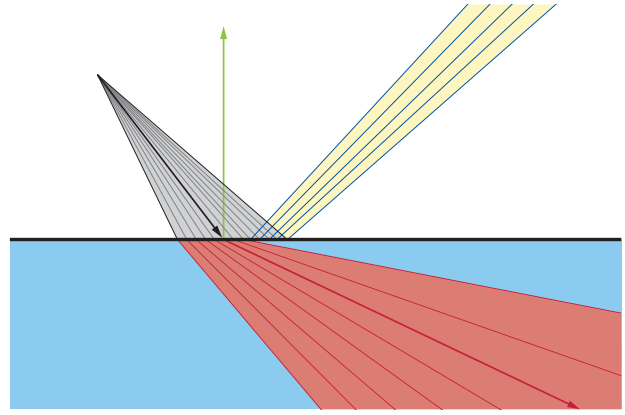

Figure 10-3. *A ray cone (in gray) is transported via a surface interface from a thicker medium to a thinner one. The central ray of the ray cone is refracted into the red arrow. Note, however, that if the ray cone is seen as a set of internal rays, four of these will be totally internally reflected, which forms the yellow part. The rest are refracted into the red volume.*

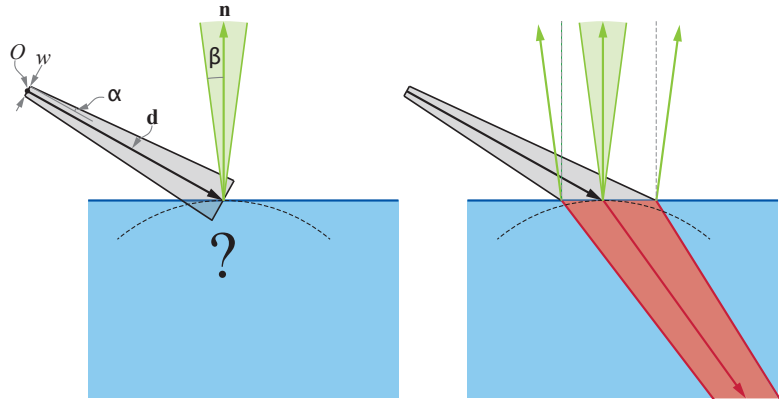

Figure 10-4. *Left: a gray ray cone, defined by an origin O, a direction* **d**, *a width w, and a half cone angle α. The spread of the normal* **n** *is modeled by the angle β. Right: we create our refraction ray cone approximation from the red volume. Dashed lines indicate the curvature of the surface at the hit point.*

a convex surface and a negative sign indicates a concave surface. Our assumption is that a reasonable approximation will be obtained by handling the computations in two dimensions, by tracking the lower and upper parts of the ray cone, and by refracting those with the perturbed normals as shown in Figure 10-4, right.

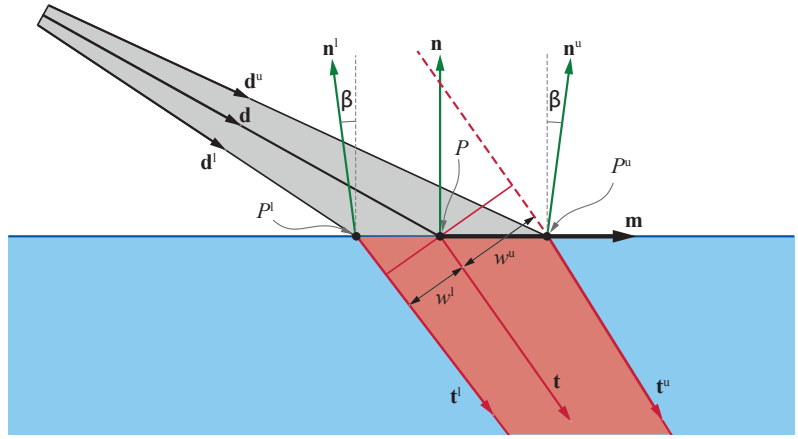

Figure 10-5. *The geometric setup used in the derivation of the refracted ray cone.*

In the following, all references are made to Figure 10-5. We use the hit point P as the origin of the two-dimensional coordinate frame, and we use $\mathbf{n}$ as the y-axis and $\mathbf{m}$ as the x-axis. The vector $\mathbf{m}$ needs to be orthogonal to $\mathbf{n}$ and also parallel to $\mathbf{d}$, and this can be done with a projection and normalization as

$$\mathbf{m} = \frac{\mathbf{d} - (\mathbf{n} \cdot \mathbf{d})\mathbf{n}}{\|\mathbf{d} - (\mathbf{n} \cdot \mathbf{d})\mathbf{n}\|}. \tag{10.1}$$

We assume that all vectors and points are in this two-dimensional frame. We create the *upper* direction vector $\mathbf{d}^u$ by rotating $\mathbf{d}$ by $+\alpha$, and the *lower* direction vector $\mathbf{d}^l$ by rotating $\mathbf{d}$ by $-\alpha$. The upper ray, with direction $\mathbf{d}^u$, starts from the origin O offset by $w/2$ in the direction orthogonal to $\mathbf{d}$, and similar for the lower ray. These are then intersected with the x-axis, which gives us P^u and P^l. This is illustrated in Figure 10-5.

We use an augmented refraction function that returns a vector located in the plane being spanned by the incident ray direction and the normal, and also orthogonal to the normal, if the ray is TIRed. Otherwise, standard refraction is used. For example, if $\mathbf{d}$ would have been TIRed in Figure 10-5, then our refraction function would return $\mathbf{m}$. From the spread of the normal and the normal itself, $\mathbf{n}$, we also create $\mathbf{n}^u$ and $\mathbf{n}^l$, which are shown in Figure 10-5. The refracted direction $\mathbf{t}$ is computed by calling the refraction function with $\mathbf{n}$ and $\mathbf{d}$ and the ratio of indices of refraction. This is done analogously to create $\mathbf{t}^u$ and $\mathbf{t}^l$ using $\mathbf{n}^u$ and $\mathbf{n}^l$.

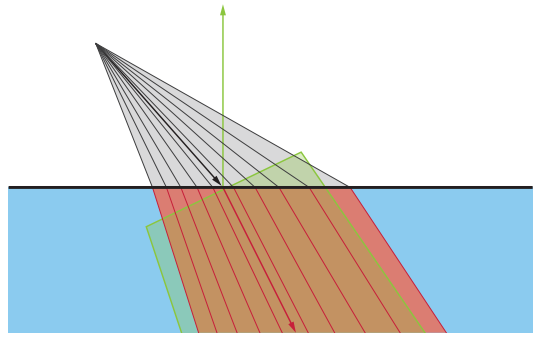

Figure 10-6. *A refracted cone calculated using our method (light green) and the ground truth approximated using many rays (red).*

Given the computations above, the refracted ray cone is constructed as follows. The refracted ray cone origin is P and its direction is $\mathbf{t}$. The width w of the refracted ray cone is computed as

$$w = w^{\mathrm{u}} + w^{\mathrm{l}}, \tag{10.2}$$

where w^{u} and w^{l} are the widths shown in Figure 10-5. The width w^{u} is computed as the length along a direction that is orthogonal to $\mathbf{t}$ from the origin to the line defined by $\mathbf{t}^{\mathrm{u}}$ and the hit point P^{u}, and analogously for w^{l}. The half cone angle α of the refracted ray cone is computed as half the angle between $\mathbf{t}^{\mathrm{u}}$ and $\mathbf{t}^{\mathrm{l}}$ together with a sign as

$$\alpha = \frac{1}{2} \arccos\left(\mathbf{t}^{\mathrm{u}} \cdot \mathbf{t}^{\mathrm{l}}\right) \operatorname{sign}\left(t_x^{\mathrm{u}} t_y^{\mathrm{l}} - t_y^{\mathrm{u}} t_x^{\mathrm{l}}\right). \tag{10.3}$$

In order to further improve the quality of our method, we suggest using higher-quality anisotropic filtering [1] for hits after refraction has occurred. The ray cone geometry changes significantly after refracting, as described in Section 10.1 and illustrated in Figure 10-6. To compensate for the imperfect approximation of the refracted cone, we can use the anisotropic filter to at least better match the elliptical footprint at the hit point.

One situation that is not explicitly handled in our method is when the incident ray is below the perturbed normal of the surface. In this case, a reflection should occur instead of refraction. However, during our experiments, we did not detect this situation happening, and without handling it explicitly, our implementation causes the cone to grow instead of reflecting. To handle this situation, we propose to clamp incident vectors to be at most 90° away from the normal.

Our method cannot only be used for perfect refraction, but also for rough refractions when, e.g., a microfacet-based BTDF is used to generate a randomized refracted direction based on surface roughness. In this case, the half-vector used for refracting the incident direction should be used as the normal for our method because it is, in fact, the normal of the microfacet that is used to refract the ray. Methods for stochastic evaluation of microfacet BTDF typically generate these half-vectors, which can be used directly. Alternatively, we can calculate such a half-vector using

$$\mathbf{n} = -\frac{\eta\mathbf{t} + \mathbf{d}}{\|\eta\mathbf{t} + \mathbf{d}\|}. \tag{10.4}$$

10.3 RESULTS

To evaluate our method, we compare the results to the ground-truth reference that stochastically samples the correct footprint and to the method based on ray differentials [8]. In Figure 10-7, we show results from a Whitted ray tracer with several different texture filtering methods, namely, accessing only mipmap level 0 (no filtering), using ray cones with isotropic filtering [2], using ray cones with anisotropic filtering [1], and using ray differentials with both isotropic and anisotropic filtering [8].

Our test scene contains a reflective sphere, a solid glass cube, and a glass sphere, where the latter two cover refraction under many angles in one view and also the case of total internal reflection. However, none of the methods explicitly handle the boundary between refraction and TIR (which is usually one or two pixels wide), and only the ground-truth method can render it correctly—the other methods produce a boundary that is too sharp. Visual results of our ray cone method are mostly comparable to the ray differentials (see Figure 10-7). Our method, however, requires less per-ray storage and overhead. Note that, as mentioned previously, we prefer to use anisotropic filtering for refraction, which also gives results closer to the ground truth.

Performance of our method is nontrivial to evaluate, as opposed to when only reflection occurs, as our new method is only used for rays hitting transmissive materials. We used a test scene containing glass objects and changed the view so that most of the scene is seen through the lens. Results are summarized in Figure 10-8. Performance of all methods is near identical, but slower than simply accessing mip level 0, which is expected. (The rendering times for the viewpoint shown in Figure 10-1 with path tracing at 4 spp are 76.8 ms for mip level 0, 75.4 ms for ray cones with isotropic filtering,

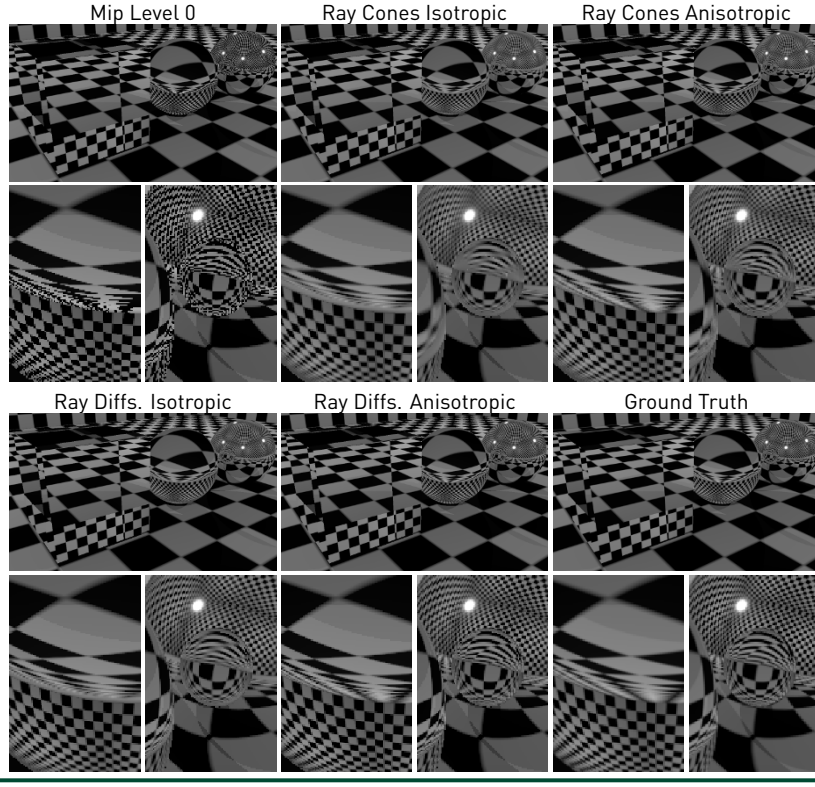

Figure 10-7. *Results for our refracted ray cone method, which used the Unified method [1] for computing surface spread. The ground truth was generated by stochastically sampling the direction inside the screen-space ray cone through a pixel with 10,000 samples per pixel. Note that anisotropic ray cones generate results that are close to anisotropic ray differentials in the left zoom-in, whereas in the right zoom-in, the ray cones are more blurry. This is because ray cones can only model circular cones and this situation requires more flexibility than that. Ray differentials have a more expensive representation, which allows for better results.*

82.7 ms for ray differentials with isotropic filtering, and 84.9 ms for ray differentials with anisotropic filtering.) The isotropic versions are faster than the anisotropic ones, but the method when we only use anisotropic filtering after a refraction event has occurred can be expected to be faster than anisotropic filtering for all hits in demanding scenes.

We also validated our approach inside a path tracer that stochastically generates path samples that integrate over both the camera pixel footprint (antialiasing) and a thin lens aperture (providing a defocus blur effect). We adapted the ray cone creation procedure in order to better match the *double cone* footprint of the light beams generated by such a setup. We take

Figure 10-8. *The Marbles scene rendered at* 1920 × 1080 *resolution with our path tracer. Left: for reference, the picture rendered using full-resolution textures (mip level 0). Right: our result using ray cones with isotropic filtering produces visually identical results. The viewpoint in the top row was rendered using 1000 spp (samples per pixel) for both approaches, and the bottom row uses 400 spp.*

advantage of the standard functionality of the ray cones and create them such that they start from the camera at aperture width, converge to zero at the focal distance, and then grow again. This allows our method to access mip levels with lower resolution even more in the blurry regions of the defocus blur. In such a path tracing setup, ray cones allow for saving bandwidth during texture sampling operations, resulting in ca. 1.5% full-frame performance improvement in the scene presented in Figure 10-1 and Figure 10-8. On this scene, our approach also renders ca. 10% faster than ray differentials with isotropic filtering and ca. 12% faster than ray differentials with anisotropic filtering. As it provides the path tracing integrator with prefiltered bandlimited texture samples, our approach also improves image convergence, as can be seen in the low-sample count results presented in Figure 10-9. Those pictures were rendered at interactive frame rates (62 ms per frame) after three accumulated frames of a static camera, using 1 spp per frame (resulting in 3 spp effective) and denoised using the NVIDIA OptiX denoiser.

Our tests were performed using an NVIDIA RTX 3090 GPU (Ampere). Compared to always accessing the finest mip level (0), ray cones with mipmapping can achieve speedups in general. This is due to accessing mip levels with lower resolution more often. However, speedups can be expected

Figure 10-9. *Interactive path tracing after three accumulated frames with low sample count (1 spp/frame resulting in 3 spp effective) using screen-space denoising and rendered at* 1920×1080*. Our approach (bottom insets) provides smoother outputs in the blurry regions compared to sampling textures at maximum resolution (mip level 0, top insets), which can help with providing faster convergence speeds.*

only for the most demanding scenes with a very large number of high-resolution textures. We also used uncompressed textures for those experiments, and enabling texture compression can reduce those speedups.

10.4 CONCLUSION

With the growing use of high-resolution textures and ray tracing in games, it is important to support a precise solution for their filtering in order to preserve the fine detail meticulously created by artists. In this chapter, we have continued development of texture filtering with ray cones and added support for refraction. Building on top of previous work, ray cones can now support a range of use cases including reflection, refraction, isotropic or anisotropic filtering, shading level of detail, integration with or without G-buffer, and more. Ray cones are a lightweight and relatively easy-to-integrate method, and we provide a publicly available implementation as part of the Falcor framework [4] (see the `RayFootprint` class).

REFERENCES

[1] Akenine-Möller, T., Crassin, C., Boksansky, J., Belcour, L., Panteleev, A., and Wright, O. Improved shader and texture level of detail using ray cones. *Journal of Computer Graphics Techniques (JCGT)*, 10(1):1–24, 2021. http://jcgt.org/published/0010/01/01/.

[2] Akenine-Möller, T., Nilsson, J., Andersson, M., Barré-Brisebois, C., Toth, R., and Karras, T. Texture level-of-detail strategies for real-time ray tracing. In E. Haines and T. Akenine-Möller, editors, *Ray Tracing Gems*, chapter 20, pages 321–345. Apress, 2019.

[3] Belcour, L., Yan, L.-Q., Ramamoorthi, R., and Nowrouzezahrai, D. Antialiasing complex global illumination effects in path-space. *ACM Transactions on Graphics*, 36(4):75b:1–75b:13, 2017. DOI: 10.1145/3072959.2990495.

[4] Benty, N., Yao, K.-H., Foley, T., Kaplanyan, A. S., Lavelle, C., Wyman, C., and Vijay, A. The Falcor real-time rendering framework. https://github.com/NVIDIAGameWorks/Falcor, 2017.

[5] De Greve, B. Reflections and refractions in ray tracing. https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf, 2006.

[6] Elek, O., Bauszat, P., Ritschel, T., Magnor, M., and Seidel, H.-P. Progressive spectral ray differentials. *Computer Graphics Forum*, 33(4):113–122, Oct. 2014. DOI: 10.1111/cgf.12418.

[7] Heitz, E. Sampling the GGX distribution of visible normals. *Journal of Computer Graphics Techniques*, 7(4):1–13, 2018. http://jcgt.org/published/0007/04/01/.

[8] Igehy, H. Tracing ray differentials. In *Proceedings of SIGGRAPH 99*, pages 179–186, 1999.

[9] Stam, J. An illumination model for a skin layer bounded by rough surfaces. In *Eurographics Workshop on Rendering*, pages 39–52, 2001.

[10] Walter, B., Marschner, S. R., Li, H., and Torrance, K. E. Microfacet models for refraction through rough surfaces. In *Rendering Techniques*, pages 195–206, 2007.

Ray Tracing Gems II

Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX

Edited by
Adam Marrs, Peter Shirley, and Ingo Wald

Section Editors
Per Christensen
David Hart
Thomas Müller
Jacob Munkberg
Angelo Pesce
Josef Spjut
Michael Vance
Cem Yuksel

Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX

Edited by
Adam Marrs
Peter Shirley
Ingo Wald

Section Editors
Per Christensen
David Hart
Thomas Müller
Jacob Munkberg
Angelo Pesce
Josef Spjut
Michael Vance
Cem Yuksel

ISBN-13 (pbk): 978-1-4842-7184-1
ISBN-13 (electronic): 978-1-4842-7185-8
https://doi.org/10.1007/978-1-4842-7185-8

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editors: Susan McDermott, Natalie Pao
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Cover image designed by NVIDIA

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484271841. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper