Differentiable Shadow Rendering

Dave Pagurek and Jerry Yin
The University of British Columbia
{dpagurek,jerryyin}@cs.ubc.ca

1. Introduction

We propose a differentiable formulation for soft shadows using shadow mapping, and implement it in a differentiable renderer. Our implementation enables applications such as optimizing a model with respect to its shadow. An example task motivating this work is the algorithmic generation of shadow art, in which a 3D shape produces different artist-specified shadow shapes when lit from different angles. An example of shadow art can be found on the cover of the book *Gödel, Escher, Bach* [4]. Our goal is to find a reasonable tradeoff in this optimization task between visual quality and performance.

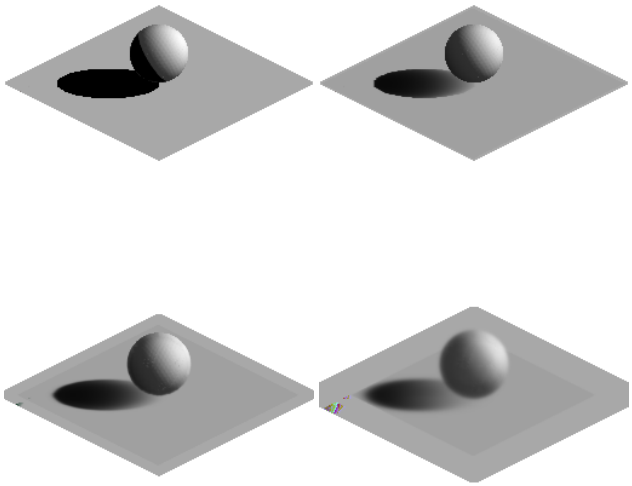

Figure 1. Effect of increasing softness parameters when rendering hard shadows in the forward pass using our differentiable formulation. The top left image was rendered with $\gamma_s = 0.01$, $\beta = 50$, $\sigma_s = 0.5$, $\sigma = 10^{-5}$, and $\gamma = 10^{-4}$. The rest used $\gamma_s = 2/3$, $\beta = 3$, and $\sigma_s = 2 \times 10^4 \sigma$ with σ values set to $10^{-5}, 10^{-4}, 10^{-3.2}$ respectively, and γ values set to $10^{-4}, 10^{-3}, 10^{-2.2}$ respectively.

2. Related work

2.1. Shadow rendering

One of the first methods [13] of rendering shadows, still used in real time applications today for its efficiency, is *shadow mapping*. Shadow mapping accounts for the occlusions of direct light from a point source. This involves rendering a depth buffer, an image recording the depth of the closest object at each pixel, from the perspective of a light. To then check if a pixel is in shadow or not, one checks whether or not the distance from the fragment at that pixel to the light is further than the distance recorded in the depth buffer.

Due to using point light sources, shadow mapping casts hard, binary shadows, as each fragment of the scene either has a direct line of sight to the light source or it does not. Soft shadows from direct illumination occur as a result of having area lights, where fragments may have direct, unoccluded lines to some but not all points on the surface of the light source. Random sampling of these points produces a shadow penumbra [2], converging to the exact penumbra, but this is expensive to do for each fragment in the scene. Algorithms to combine multiple shadow maps sampled from the surface of a light exist [3], but can still suffer from performance issues when enough samples are used to get reasonably accurate estimates.

Other methods make assumptions to approximate light sources and generate soft shadows more efficiently. If light sources are spheres, the amount of occlusion of the cone from a fragment to the light source can be used to create a shadow penumbra [1]. If, in addition to a shadow map, one knows the closest distance from an almost-occluder to the line from a fragment to the center of a light source, this distance can be used to approximate the cone occlusion [11]. Due to the minimal amount of information required and the smooth shadows it produces, our smooth shadows are based off of this method.

2.2. Shadow optimization

The problem of optimizing a model to produce a set of shadows has been examined previously [9] using a voxel-based model and simple lighting. In this formulation, shad-

ows are projected back onto the voxel grid, deleting voxels that are outside the shadow volume. The resulting voxel grid is then distorted to balance discrepancies introduced by the voxels removed by other shadow angles.

2.3. Differential rendering

The more general task of taking the derivative of an image with respect to scene parameters is called *differentiable rendering*. Many methods ignore shadows, focusing instead on calculating approximate occlusion derivatives at edges [8, 5], using volume distributions [12], or using probabilistic interpretations of mesh polygons [7]. Other methods attempt to make the whole physically-based rendering pipeline differentiable [6, 10], supporting accurate shadows at the cost of added complexity and longer render times to reach an image that is reasonably free of noise.

3. Method

We add an approximate shadowing model to Soft Rasterizer [7], an existing differentiable rendering system which does not use a Monte Carlo approach.

3.1. Hard shadows

We begin by implementing a point light model, which casts hard shadows. Traditionally, hard shadow mapping is done in two passes: in the first pass, a shadow map image is computed from the perspective of the light source, treating the light source as a separate "camera," then the depth of the closest fragment at each pixel is stored. Then, in the second pass where the scene is rendered from the perspective of the scene camera, each fragment being shaded determines whether or not it is in shadow by checking if it is visible from the light's perspective. This can be done by transforming the position of the fragment being shaded into the coordinate frame of the light camera, then sampling the corresponding point from the shadow map to determine the depth of the closest point that is hit along that line of sight. If the closest point is closer than the fragment being shaded, then the fragment being shaded is obscured by that point and thus should be in shadow.

The depth map computation would not be differentiable if we merely stored the closest depth. Instead, we store in the shadow map a *soft* minimum depth over all triangles i. Like Soft Rasterizer, we define the influence of each triangle f_j at point i to be probability map

$$\mathcal{D}_j^i = \text{sigmoid}\left(\delta_j^i \cdot \frac{d^2(i,j)}{\sigma}\right), \tag{1}$$

where σ is the sharpness parameter, δ_j^i is a sign indicator ($+1$ for points inside the face, -1 otherwise), and $d(i,j)$ is some distance metric to f_j's edges. Our soft depth formulation is similar to Soft Rasterizer's soft formulation for

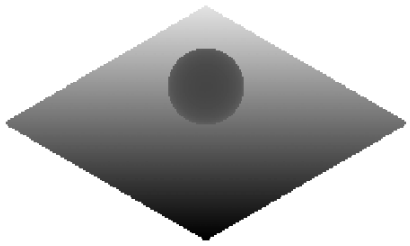

Figure 2. Visualization of the depth map.

aggregating colour C,

$$I^i_{\text{colour}} = \mathcal{A}_S(\{C_j\}) = \sum_j w_j^i C_j^i + w_b^i C_b, \tag{2}$$

with weights $\sum_j w_j^i + w_b^i = 1$ and

$$w_j^i = \frac{\mathcal{D}_j^i \exp(1/(z_j^i \gamma))}{\sum_k \mathcal{D}_k^i \exp(1/(z_k^i \gamma)) + \exp(1/(\epsilon\gamma))}, \tag{3}$$

except we aggregate the depth values z_j^i instead of colour:

$$I^i_{\text{depth}} = \mathcal{A}_S(\{z_j\}) = \sum_j w_j^i z_j^i + w_b^i z_b. \tag{4}$$

The parameter γ controls the sharpness of the aggregation, z_b defines the depth of the background, which should be a very large value, and ϵ controls the effect of the background. Figure 2 shows the depth map from the perspective of the light that was used to render the top left image of Figure 1.

During the second pass, merely sampling a single pixel from the shadow map would also not be differentiable with respect to the location being sampled. Therefore, we compute a weighted sum of all pixels in the shadow map, with weights corresponding to a 2D Gaussian distribution centred around the sampling point, normalized to sum to one. Thus a shadow map sample from location (μ_x, μ_y) on map S is

$$\text{sample}(S, \mu_x, \mu_y) = \sum_{0 \le i < W} \sum_{0 \le j < H} \lambda_{i,j} S[i,j], \tag{5}$$

where

$$\lambda_{i,j} = \frac{\exp\left(-\frac{(\mu_x - i)^2 + (\mu_y - j)^2}{2\sigma_s}\right)}{\sum_{0 \le i' < W} \sum_{0 \le j' < H} \exp\left(-\frac{(\mu_x - i')^2 + (\mu_y - j')^2}{2\sigma_s}\right)} \tag{6}$$

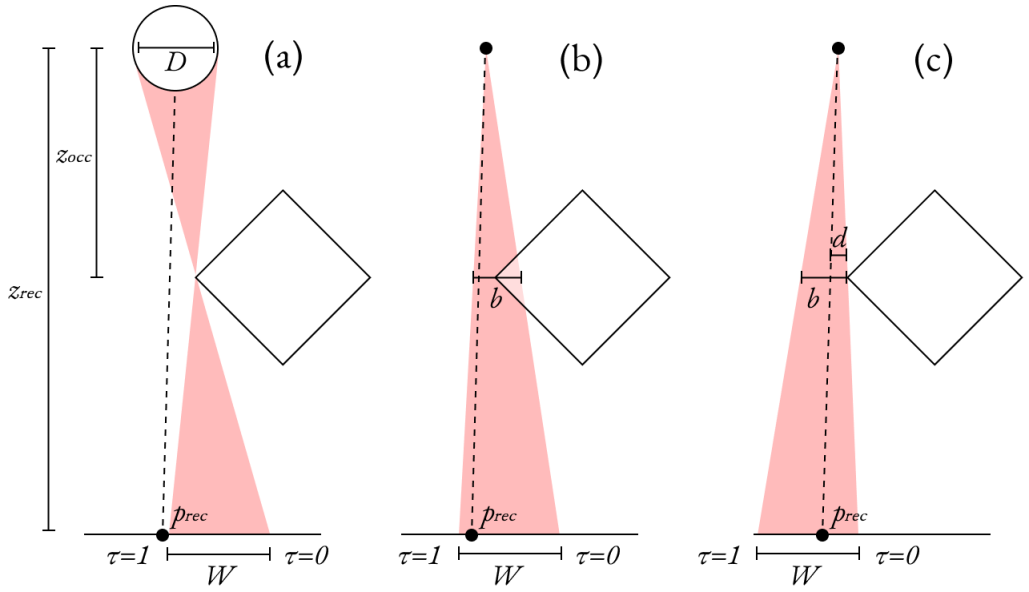

Figure 3. Point light sources are approximated as a sphere light with diameter D, whose true penumbra has a width W (a). The same penumbra size is used to create a cone representing the penumbra, with both inner and outer penumbras (b), and with just an outer penumbra (c).

and W, H are the width and height of the shadow map. In the above form, it takes $O(WH)$ time to compute a single sample, however this process can be reduced to an $O(1)$ one at the cost of slightly less accurate gradients by truncating all $\lambda_{i,j}$ below a certain threshold to 0 (assuming σ_s is fixed).

The value of σ_s when defining the weights $\lambda_{i,j}$ is allowed to differ from the sharpness parameter σ; this σ_s merely controls the effect of pixels close to the sampled point. As $\sigma_s \to 0$, our sampling formulation becomes equivalent to traditional one-hot sampling. Note that our formulation becomes differentiable with respect to the sampling point because μ_x and μ_y need not be aligned with any pixel.

There is one last computation in traditional shadow mapping which is not differentiable, which is the depth comparison between the shaded fragment and the depth value sampled from the map. Instead of a hard comparison, we use a sigmoid function, which has the benefit of having a non-zero derivative everywhere. Let d_j^i denote the distance from the shaded fragment to the light. Our differentiable shadow term is

$$s_j^i = \text{sigmoid}\left(\frac{1}{\gamma_s}(\text{sample}(S, \mu_x, \mu_y) - d_j^i) + \beta\right), \quad (7)$$

where γ_s controls the softness of the comparison, and β is a bias value which can be used to adjust the value of the shadow term for surfaces not in shadow. s_j^i is close to 1 for fragments not in shadow, and is close to 0 for fragments in shadow. Note that when $\gamma_s \to 0$, the shadow term becomes equivalent to a traditional hard shadow term, and the role of β becomes that of the bias term which is commonly used to prevent shadow acne caused by self-shadowing (which in turn results from numerical imprecision). Again, γ_s is allowed to differ from the aggregation sharpness term γ.

In the end, the shadow term is then multiplied with the direct lighting term, yielding the final colour

$$C_{j,\text{total}}^i = s_j^i C_{j,\text{direct}}^i + C_{\text{ambient}}. \quad (8)$$

3.2. Soft shadows

Soft shadows due to direct light sources occur when a light source has finite surface area that can be partially occluded. The area in full shadow, the *umbra*, is the region with no unoccluded line to any of the surface area of the light. The area in partial shadow, the *penumbra*, has a view of some, but not all, of the light. Areas with no shadow have an unobscured view of the whole light. Monte Carlo renderers create soft shadows by averaging over multiple paths from a fragment to different sampled locations on the surface of the light. Real-time, deterministic soft shadows create this effect by approximating the level of partial occlusion at each fragment.

We approximate the penumbra region using an existing technique where one constructs a cone between the light source and receiving fragment, where one end of the cone is the outer edge of the umbra, and the other edge of the cone is the outer edge of the penumbra [11]. The amount of direct light received, which we refer to as $\tau \in [0, 1]$, is found by checking where in the penumbra cone a fragment falls. We define $\tau = 0$ at the inner edge of the cone, and $\tau = 1$ at the outer edge.

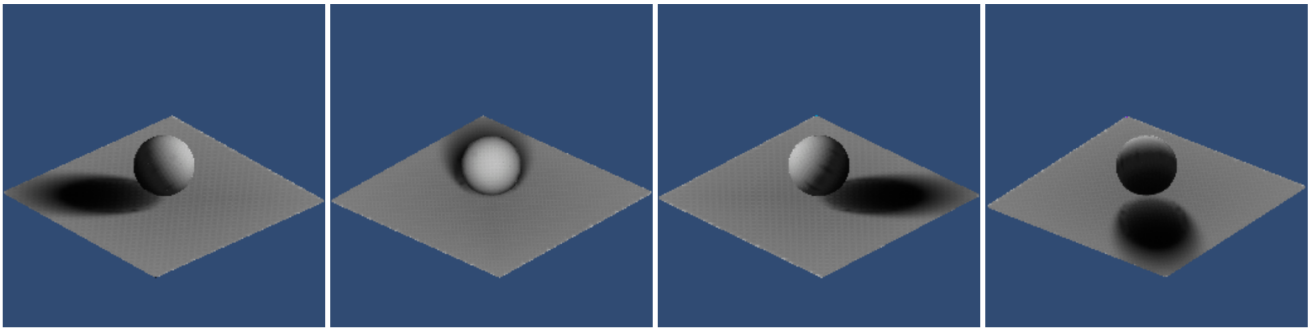

Figure 4. Frames from the forward pass of our differentiable soft shadow renderer.

The cone size is picked such that the penumbra size W at the receiver produced by a true sphere light, shown in Figure 3a, is maintained in the approximation, shown in Figure 3b and c. The paper introducing this technique derived that the diameter W is maintained at the receiver when the cone diameter at the occluder is given the value $b = \frac{z_{rec} - z_{occ}}{z_{rec}}$. To produce a penumbra that extends both into and out of the hard shadow region, the penumbra cone is centered at the edge of the occluder (Figure 3b). To only extend the hard shadow region without intruding into it, the edge of the occluder is used as the edge of the cone (Figure 3c). We choose to extend the hard shadow region (an "outer penumbra"), as this allows a shadow map to be used to efficiently check whether a fragment is in the umbra or not.

With only an outer penumbra, τ can be calculated by measuring the distance d between the edge of the occluder and the line from the receiving point and the light using the formula $\tau = \frac{d}{b}$. When the receiving point is at the outer edge of the cone, $d = b$ and $\tau = 1$, indicating full lightness. At the inner edge of the cone, $d = 0$, so $\tau = 0$ and no direct light is received. With multiple occluders, we take τ_{min} over all occluders. To make this differentiable, we again use a soft minimum, but only over the triangles T which are closer to the light source than the receiving triangle. This value, representing the partial occlusion of the frontmost fragment to the light, is then stored in a *penumbra map* as $P[i, j]$, an example of which is shown in Figure 5.

$$P[i,j] = \frac{\sum_{t \in T_j^i} \tau_t \exp(-\tau_t \gamma)}{\sum_{t \in T_j^i} \exp(-\tau_t \gamma)}, \tag{9}$$

$$T_j^i = \{t \,|\, z_t < S[i,j] + \beta\} \tag{10}$$

Finally, to calculate the lightness s_j^i of a fragment for triangle t in the scene using soft shadows instead of hard shadows, one makes use of both the shadow map S and the penumbra map P. Like with hard shadows, Gaussian weights are used to smoothly sample the map, but where the penumbra value is multiplied by the sigmoid of the shadow value. This is interpreted as multiplying the amount of light that would be received for the frontmost fragment by the probability that the fragment in question is in front.

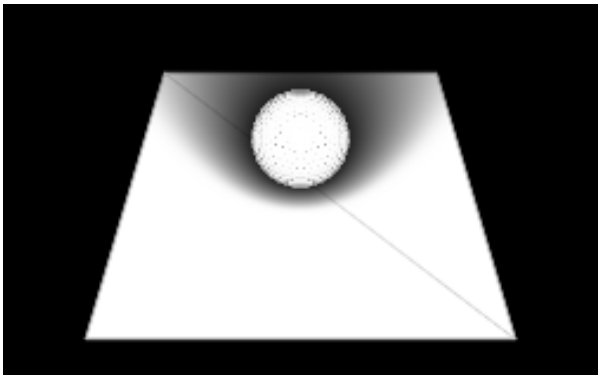

Figure 5. An example of a penumbra map, showing a soft shadow around an occluder.

$$s_{j,t}^i = \sum_{i=0}^{W} \sum_{j=0}^{H} \lambda_{i,j} P[i,j] \operatorname{sigmoid}\left(\frac{1}{\gamma_s}(S[i,j] - d_{j,t}^i + \beta)\right) \tag{11}$$

Like before, this shadow term is used as a multiplier for the direct light received at the fragment.

4. Evaluation

To test forward rendering, we rendered a 360° turnaround animation of a sphere on a plane lit by a single light. A subset of the frames from this animation is shown in Figure 4. It took 1m44s to render the whole animation, consisting of 90 frames. Rendering the same scene using only hard shadows takes 33s, and rendering without shadows takes 1s.

To test the backwards pass, we would like to optimize the shape of an object based on the shadow it casts. If one were to optimize shape based on its hard shadow on a flat surface, this can be tested indirectly by placing a camera at the location of the light source and pointing it at the object in question and looking at the silhouette produced. We

Figure 6. Top: target silhouettes for the front and side view of an object. Bottom: the optimized silhouettes of the object.

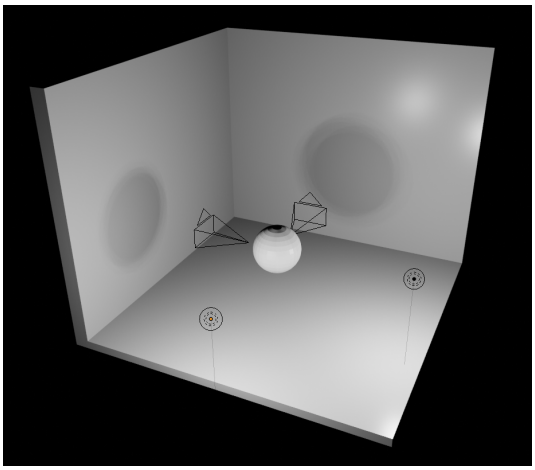

Figure 7. The scene setup used for gradient evaluation.

use this test as a performance baseline. Using two cameras aimed at the front and side of a sphere, we use intersection-over-union loss between the captured silhouettes and target silhouettes we provide to optimize the vertex positions of the sphere. The result of this optimization is shown in Figure 6.

If shadows are not hard or if the camera is placed at an angle to a non-flat surface, the previous setup will not work. To test the ability to optimize a shape based on its shadows directly, we created the test scene shown in Figure 7. In the center is a shape we wish to optimize, which begins as a sphere. Lights are placed on two perpendicular sides of the shape. For each light, a camera is placed between the shape and the wall so that the image from the camera will be of the shape's shadow. Target images are provided that each camera image will be compared to using L2 loss. Gradients from the sum of these losses are used to adjust the vertex positions of the sphere. Note that while this scene setup does not use non-flat surfaces or non-perpendicular camera angles, the same underlying shadow gradients must exist to enable optimization.

Due to time constraints, we were not able to complete our implementation of the gradients for soft shadows, so while the test runner script is in place, we leave running this test for future work.

5. Discussion

We notice that inclusion of hard and soft shadows has a significant effect on render time. We believe this is an unavoidable cost of rendering a smoothly differentiable image. If one removes the sigmoid function when comparing the depth of a fragment to the value stored in the depth map and instead use hard 0-or-1 occlusion, the render time of the animation if Figure 4 reduces to a third of its previous render time. Removing the Gaussian smoothing when sampling from the depth and penumbra maps reduces the render time to mere seconds. The information locality that makes traditional shadow rendering efficient necessarily must be removed in order to have gradients that account for small changes that might result in different map samples being used.

While we were able to produce images that look visually correct, we were not able to test how well our differentiable soft shadows work for use in optimization. This is largely due to the fact that Soft Rasterizer is implemented as a custom CUDA kernel, meaning that gradients must be manually computed. Future work includes the completion of our partially-implemented backwards pass. To test that it is implemented correctly, we would like to compare our computed gradients with approximated gradients using finite differences of forward pass values. After ensuring that the gradients are correct, we would then like to run the shadow optimization test described in the Evaluation section.

References

[1] John Amanatides. Ray tracing with cones. *SIGGRAPH Comput. Graph.*, 18(3):129–135, Jan. 1984.

[2] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, Jan. 1984.

[3] Michael Herf and Paul S. Heckbert. Fast soft shadows. In *ACM SIGGRAPH 96 Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH '96*, SIGGRAPH '96, pages 145–, New York, NY, USA, 1996. ACM.

[4] Douglas R. Hofstadter. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc., USA, 1979.

[5] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[6] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte Carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 37(6), 2018.

[7] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning, 2019.

[8] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *Computer Vision –*

ECCV 2014, volume 8695 of *Lecture Notes in Computer Science*, pages 154–169, Sept. 2014.

[9] N. J. Mitra and M. Pauly. Shadow art. *ACM Trans. Graph.*, 28(5), 2009.

[10] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 38(6), Dec. 2019.

[11] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical report, 1998.

[12] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proc. 2015 ICCV*, 2015.

[13] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, page 270–274, New York, NY, USA, 1978. Association for Computing Machinery.