Distributions for Compositionally Differentiating Parametric Discontinuities

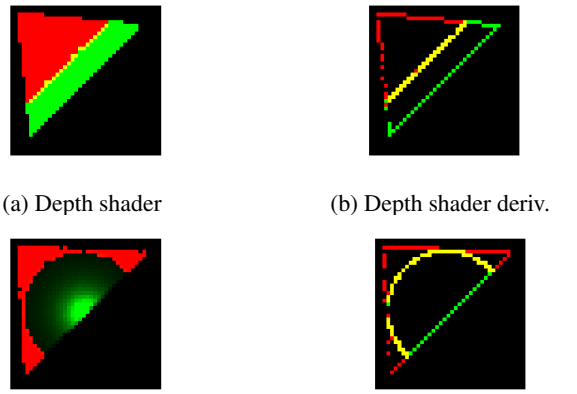
Jesse Michel [1] **Kevin Mu** [2] **Xuanda Yang** [3] **Sai Praveen Bangaru** [1] **Elias Rojas Collins** [1] **Gilbert Bernstein** [2]
Jonathan Ragan-Kelley [1] **Michael Carbin** [1] **Tzu-Mao Li** [3]

Abstract

Computations in computer graphics, robotics, and probabilistic inference often require differentiating integrals with discontinuous integrands. Popular differentiable programming languages do not support the differentiation of these integrals. To address this problem, we extend distribution theory to provide semantic definitions for a broad class of programs in a programming language, Potto. Potto can differentiate parametric discontinuities under integration, and it also supports first-order functions and compositional evaluation. We formalize the meaning of programs using denotational semantics and the evaluation of programs using operational semantics. We prove correctness theorems and show that the operational semantics is compositional, enabling separate compilation and overcoming compile-time bottlenecks. Using Potto, we prototype a differentiable renderer with separately compiled shaders.

1. Introduction

The advent of deep learning frameworks supporting automatic differentiation—the automated computation of derivatives of a function given just the definition of the function itself—within general-purpose programming languages has opened up new avenues to build large-scale applications for differentiable programming, including in deep learning, optimization, and uncertainty quantification. However, these techniques have traditionally been limited to continuous processes, excluding a variety of natural phenomena that are typically modeled as discontinuous functions.



(a) Depth shader (b) Depth shader deriv.

(c) Thresh. Lambert shader (d) Thresh. Lambert deriv.

Figure 1. Potto can separately compile and swap between different shaders, making the design process tractable. The sparsity pattern of the derivative shaders are displayed—the derivative shader is displayed red, say, if the derivative is nonzero in the red channel.

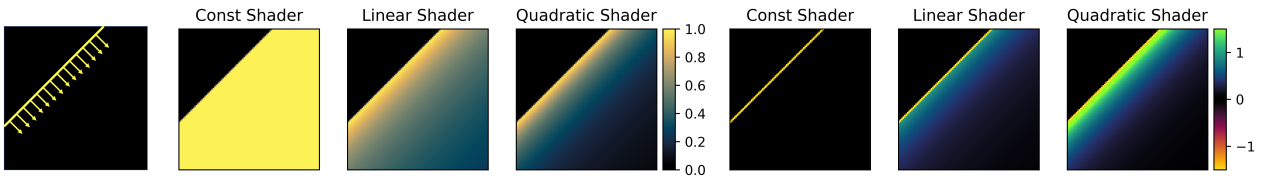
Such discontinuous functions arise in computer graphics and vision (Li et al., 2018), robotics (Hu et al., 2020; Bangaru et al., 2021), and probabilistic inference (Lee et al., 2018). In computer graphics, discontinuities arise from object boundaries, occlusion, and sharp changes in color. Robot controllers and computer simulations often model contact, which introduces discontinuities. In probabilistic inference, the models often have discontinuities, e.g., a probabilistic simulation of a controller that regulates room temperature is discontinuous due to switching a heater on/off.

Consider a program that models contact, such as a simulation or animation of a robot walking (Stengel, 1994; Witkin & Kass, 1988). Such a program might compute the integral $\int_0^1 [x < \theta]\, dx$, where the Iverson bracket $[P]$ is one if P is true and zero otherwise. The parameter θ represents the time the ball hits the wall and $x < \theta$ is a *parametric discontinuity*. Although the derivative with respect to θ is $D_\theta \int_0^1 [x < \theta]\, dx = [0 < t < 1]$, popular differentiable programming languages return zero because they do not account for parametric discontinuities (Bradbury et al., 2018; Paszke et al., 2019).

Teg (Bangaru et al., 2021) automatically differentiates parametric discontinuities, providing an unbiased estimation of

*Equal contribution [1]CSAIL, MIT, 32 Vassar Street in Cambridge, Massachusetts 02139, USA [2]Computer Science and Engineering, University of Washington, University of Washington, 3800 E Stevens Way NE, Seattle, WA 98195, USA [3]Computer Science and Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. Correspondence to: Jesse Michel <jmmichel@csail.mit.edu>.

Published at the Differentiable Almost Everything Workshop of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. July 2023.



(a) Scene (b) Shaders change how light attenuates (c) The derivative w.r.t. light position

Figure 2. We implement a differentiable renderer in Potto and render a scene depicting a single line of light pointing diagonally downward (a) with different light attenuations (b). Swapping between different shaders is an example of how designers modify a scene for an artistic effect. In Potto, programs and their derivatives (c) can be separately compiled and composed to efficiently swap among shaders.

a program with integrals. However, their approach requires *global program transformations*, leading to long compile times. Our programming language, Potto, avoids these problems leading to an 81x and 441x speedup in compile time in an application to image stylization.

Moreover, global program transformations prevent *separate compilation*—the ability to compose compiled code snippets. For a renderer, this means that when an artist tweaks the scene, such as changing the color of an object (swap between shaders), the whole scene must be recompiled.

Figure 1 shows the result of using a renderer implemented in Potto to swap between two shaders applied to the same triangle in 3D space. Both the triangle and shaders can be separately compiled and composed in Potto, but not in Teg.

2. Related Work

Most automatic differentiation methods ignore discontinuities, such as if-else branches, during differentiation (Paszke et al., 2019; Bradbury et al., 2018). Normally, doing so is correct almost everywhere (Lee et al., 2020). However, when the problem specification involves integration, as in computer graphics, robotics, and probabilistic inference, ignoring the discontinuities produces incorrect gradients. In practice, these incorrect gradients results in slower convergence or even divergence during optimization (Bangaru et al., 2021; Li et al., 2018; Lee et al., 2018).

Many works account for parametric discontinuities by hand-deriving application-specific derivatives (Loper & Black, 2014; Li et al., 2018; Hu et al., 2020; Loubet et al., 2019; Bangaru et al., 2020). Emerging research accounts for parametric discontinuities: Lee et al. (2018) proposes a solution for affine discontinuities, while Yang et al. (2022); Liu et al. (2019); Chen et al. (2019); Petersen et al. (2022) support a wider class of discontinuities but introduces bias. In contrast, Teg (Bangaru et al., 2021) directly samples discontinuous surfaces that arise from differentiation, providing an unbiased, low-variance integral estimation.

Subsequent work uses distributional semantics to build an equational theory, but lacks an operational semantics (Azevedo de Amorim & Lam, 2022).

3. Case Study

We motivate our language by implementing a *differentiable renderer* (de La Gorce et al., 2011; Loper & Black, 2014; Li et al., 2018; Zhao et al., 2020; Li et al., 2020). Applications, ranging from autonomous driving and robotics to CGI, use differentiable renderers to e.g. recognize the 3D shapes of cars, signs, and pedestrians, reconstruct the 3D scene for the robot to interact with, and motion capture actors' faces.

A *renderer* is a program which takes in the geometry and color of each object in the scene and outputs an image. Renderers are built out of programs called *shaders*. A *differentiable renderer* computes the change in the color of a pixel with respect to the change of a parameter, such as the location or color of an object.

Figure 2 depicts images generated by a differentiable renderer implemented in Potto. The scene contains a single object—a line of light shining diagonally downward. The color of a single pixel is the average of light within the pixel area:

$$f(c, s) = \int_0^1 \int_0^1 s(x, y, c)[x + y + c \geq 0] \, \mathrm{d}x\mathrm{d}y \quad (1)$$

We use the convention that the origin is in the top left corner and the y-axis points down. The half plane $[x+y+c \geq 0]$ is the visibility shader for the light. The function $s : \mathbb{R}^3 \to \mathbb{R}$ is the color shader for the light that depends on the point (x, y) and parameter c.

The goal is to optimize the parameter c so that the renderer generates a pixel with color a. We use gradient descent to minimize the loss function $L(c, s) = (f(c, s) - a)^2$ with derivative $D_c L(c, s) = 2(f(c, s) - a)D_c f(c, s)$, where D_c represents the partial derivative with respect to c. A differentiable renderer computes the derivative $D_c f(c, s)$.

In order to easily iterate on designs, a user should be able to efficiently replace the visibility shader and the color shader to change the shape and color of the object, respectively.

Differentiating Parametric Discontinuities We give an informal description of the differentiating parametric discontinuities and give a formal treatment in the following

section. In Figure 2, the image using the constant shader $s(x, y, c) = 1$ shows a half plane and its derivative is a line along the boundary of the half plane. Since the derivative is the change resulting from perturbing the parameter c in $f(c, s)$. The half-plane shifts diagonally downward, making the boundary the only region with a non-zero derivative. The linear shader $s(x, y, c) = (\sqrt{2}(x + y + c) + 2)^{-1}$ and the quadratic shader $s(x, y, c) = (\sqrt{2}(x + y + c) + 2)^{-2}$ have the same boundary contribution to the derivative, but also have a non-zero interior derivative.

The derivative of the renderer decomposes into the interior and boundary contributions:

$$D_c f(c, s) = \int_S \underbrace{D_c s(x, y, c)[x + y + c \geq 0]}_{\text{interior}} + \underbrace{s(x, y, c)\delta(x + y + c)}_{\text{boundary}} \, \mathrm{d}(x, y). \tag{2}$$

The Dirac delta distribution δ in the integrand can be thought of as zero everywhere and approaching infinity along the line $x + y + c = 0$. This shows up as the diagonal yellow line in the derivative of all three shaders.

Automatic Differentiation of Parametric Discontinuities A naïve implementation would discretize the integral to a sum and use automatic differentiation to compute the derivative of the discretization. The resulting program would approximate the interior term of Equation 2, but ignore the second due to the Dirac delta distribution, producing an incorrect result.

Recent work accounts for both terms by introducing an integral primitive to a differentiable programming language, Teg (Bangaru et al., 2021). Teg performs a series of code transformations such as distributing multiplication over addition, performing a global change of variables, and applying a global symbolic rule to eliminate Dirac delta terms.

This global rewriting approach fundamentally relies upon having the syntax of the whole expression. These rules are not compositional and do not allow for separate compilation. As a result, they are a barrier to performance.

For example, separately compiling shaders is critical in video games, animation, photo-editing software, and computer-generated imagery, allowing users/designers to see multiple variants of a scene or video without requiring compiling the whole scene again. Likewise, users often run a differentiable renderer with multiple shaders.

Our differentiable programming language, Potto, has an integral primitive and can separately compile programs. We implement the renderer specified in Equation 1 and demonstrate separate compilation by writing the renderer (and visibility shader) and color shaders in separate files.

The following code snippet shows a differentiable renderer:

```
1  # renderer.po
2  from half_plane import cond
3  def renderer(c, shader)
4    integral ([0,1],[0,1])
5      ((shader (x,y)) c)
6       *(((cond (x, y)) c)?1:0) d(x,y)
7  drenderer = deriv(renderer)
```

Derivatives, integrals, and discontinuities Line 2 imports the invertible, differentiable function (diffeomorphism) `\(x,y).\c.(x+y+c,x-y+c)` from the `half_plane.diffeo` file, where the `\(x, y)` notation declares an anonymous function with parameters `x` and `y`. For instance, in Python, we write `lambda x, y:...` to declare an anonymous function. It takes in variables of integration x, y and variable c, and returns an affine combination of the three. Line 4 declares the integral to be estimated (Equation 1). The first argument specifies that x, y each range from zero to one, the second argument is the integrand, and the `d(x,y)` declares that x and y are variables of integration. In Line 5, the `deriv` operator specifies the dual number derivative of `drenderer`. The derivative `drenderer` takes in pairs of an input and an infinitesimal perturbation to that input and produces a pair of the outputs for the evaluation of the renderer and its derivative.

```
1  # color_shaders.po
2  dconst_shader = deriv(\(x,y).\c.1)
3  dlin_shader = deriv(\((x,y).\c.
4      1/(sqrt(2)*(x+y+c)+2))
5  dquad_shader = deriv(\(x,y).\c.
6      1/(sqrt(2)*(x+y+c)+2)^2)
```

The three shaders are first-order functions that model how quickly light attenuates. The `const_shader` corresponds to no attenuation—intensity is invariant to the distance from the light. While in `lin_shader` and `quad_shader`, the attenuation is linear and quadratic, respectively.

```
1  # main.po
2  from color_shaders import dconst_shader,
       dlin_shader, dquad_shader
3  from renderer import drenderer
4  for dshader in [dconst_shader,
       dlin_shader, dquad_shader]:
5      print(drenderer(dc=(-2,1), dshader))
```

A differentiable renderer In `main.po`, we compose the derivative of the renderer with the derivative of each of the color shaders to compute the color of a single pixel. We evaluate the resulting expression at a base point -2 with infinitesimal 1, producing a number representing the derivative. We can use the differentiable renderer in a derivative-based optimization procedure, such as gradient descent, to find the value of c that results in a pixel that is most similar to the pixel provided.

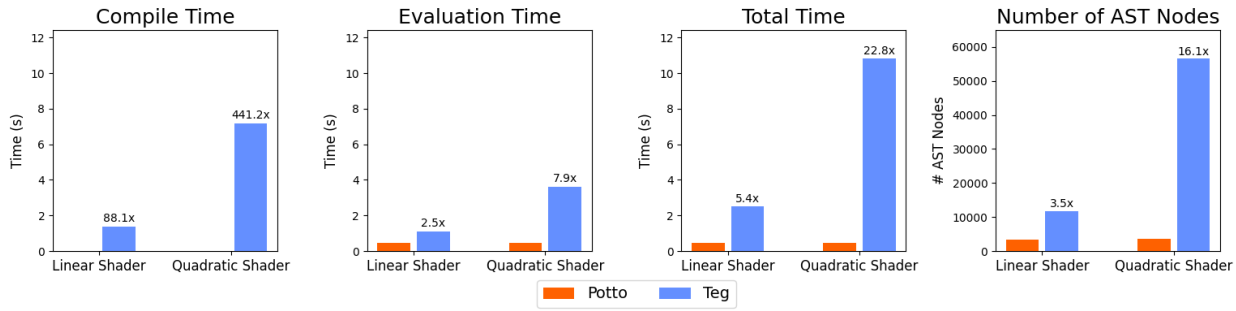


Figure 3. A bar chart, where smaller is better, comparing Potto to Teg on a rendering task for image stylization. Potto is so much faster in compile time that the bars on not visible. Compile time was the bottleneck in Teg.

Separate compilation Potto separately compiles (calculates the derivative of) the renderer and color shaders. As a result, it only compiles the renderer and each of the shaders once. In contrast, Teg must compile the renderer three times—once for each of the shaders.

4. Potto programming language

We introduce a differentiable programming language, Potto, that is the first with an integration primitive, to have first-order functions, and to support compositional evaluation. A first-order function takes in values of base types and produces values of base types. For instance, in Potto, a user can specify a function of type real to real, but not a function that that takes in another function as an argument. Compositional evaluation means that a compiler can evaluate programs independently, passing values between modules.

We present a first-order functional language for programming with distributions. The language has constants c, variables x, sums and products of arbitrary terms $t_1 + t_2$ and $t_1 \cdot t_2$, conditionals **if** t_1 **then** t_2 **else** t_3, diffeomorphic conditionals **if** $\lfloor \Psi \rfloor(x_1, \ldots)$ **then** t_1 **else** t_2, integrals **int** t **d**$(x_1, \ldots, x_n)$, pairs (t_1, t_2), and applications of first-order functions $\lfloor f \rfloor(t)$. The function Ψ is a differentiable, invertible function, taking in variables of integration and free parameters written as $(x_1, \ldots)$ and f is a piecewise differentiable function with piecewise invertible pieces. The type system (not shown) prevents variables of integration from occurring in the condition t_1 of **if** t_1 **then** t_2 **else** t_3.

Denotational semantics Building on our extension of distribution theory, we present a novel denotational semantics for programs and derivatives of programs. We prove a soundness theorem that shows that the derivative of the denotation is equivalent to the separately defined derivative denotation under mild conditions.

Operational semantics We present a novel operational semantics that we prove accords with the denotational semantics, providing unbiased estimates of the denoted program. The operational semantics is compositional and therefore supports separate compilation.

Implementation and applications We implement a prototype system, Potto, and use it to build a renderer with multiple shaders. We show that the renderer supports separate compilation, enabling interactive workflows that would otherwise be computationally intractable.

5. Empirical Results

Figure 1 depicts a differentiable ray tracing renderer of a triangle tilted in 3D space, colored using two different shaders that can be separately compiled. These *toon shaders* are 1) a *z-depth shader* that assigns color to objects based on their distance from the camera, and 2) a *thresholded Lambert shader* that models the reflectance of a matte surface under a point light in front of the triangle (Lake et al., 2000).

We compare the performance of Potto to previous work, Teg (Bangaru et al., 2021), on a rendering benchmark for image stylization. The results in Figure 3 show that Potto outperforms Teg in all areas: compile time, evaluation time, total time, and AST size. Most striking is the 88x and 441x compile time speed up of Potto relative to Teg.

6. Conclusion

In our work, we extend the scope of differentiable programming languages to handle integrals and parametric discontinuities. Potto supports compositional evaluation and as a result, transformations that were once global can be made local. This enables separate compilation. We envision that our theoretical approach and programming language design will lead to more expressive differentiable programming languages that better serve application domains including graphics, robotics, and probabilistic inference.

References

Azevedo de Amorim, P. H. and Lam, C. Distribution Theoretic Semantics for Non-Smooth Differentiable Programming. *arXiv e-prints*, 2022.

Bangaru, S., Michel, J., Mu, K., Bernstein, G., Li, T.-M., and Ragan-Kelley, J. Systematically differentiating parametric discontinuities. In *Special Interest Group on Computer Graphics and Interactive Techniques*, 2021.

Bangaru, S. P., Li, T.-M., and Durand, F. Unbiased warped-area sampling for differentiable rendering. *Special Interest Group on Computer Graphics and Interactive Techniques in Asia*, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., and Wanderman-Milne, S. JAX: composable transformations of Python+NumPy programs, 2018.

Chen, W., Gao, J., Ling, H., Smith, E., Lehtinen, J., Jacobson, A., and Fidler, S. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances In Neural Information Processing Systems*, 2019.

de La Gorce, M., Fleet, D. J., and Paragios, N. Model-based 3D hand pose estimation from monocular video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2011.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. DiffTaichi: Differentiable programming for physical simulation. *International Conference on Learning Representations*, 2020.

Lake, A., Marshall, C., Harris, M., and Blackstein, M. Stylized rendering techniques for scalable real-time 3d animation. In *International Symposium on Non-Photorealistic Animation and Rendering*, pp. 13–20, 2000.

Lee, W., Yu, H., and Yang, H. Reparameterization gradient for non-differentiable models. In *Neural Information Processing Systems*, 2018.

Lee, W., Yu, H., Rival, X., and Yang, H. On correctness of automatic differentiation for non-differentiable functions. In *Neural Information Processing Systems*, 2020.

Li, T.-M., Aittala, M., Durand, F., and Lehtinen, J. Differentiable Monte Carlo ray tracing through edge sampling. *Special Interest Group on Computer Graphics and Interactive Techniques in Asia*, 2018.

Li, T.-M., Lukáč, M., Michaël, G., and Ragan-Kelley, J. Differentiable vector graphics rasterization for editing and learning. *Special Interest Group on Computer Graphics and Interactive Techniques in Asia*, 2020.

Liu, S., Li, T., Chen, W., and Li, H. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

Loper, M. M. and Black, M. J. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*, 2014.

Loubet, G., Holzschuch, N., and Jakob, W. Reparameterizing discontinuous integrands for differentiable rendering. *Special Interest Group on Computer Graphics and Interactive Techniques in Asia*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*. 2019.

Petersen, F., Goldluecke, B., Borgelt, C., and Deussen, O. GenDR: A Generalized Differentiable Renderer. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Stengel, R. F. *Optimal control and estimation*. Courier Corporation, 1994.

Witkin, A. and Kass, M. Spacetime constraints. *Special Interest Group on Computer Graphics and Interactive Techniques*, 1988.

Yang, Y., Barnes, C., Adams, A., and Finkelstein, A. Aδ: Autodiff for discontinuous programs - applied to shaders. In *Special Interest Group on Computer Graphics and Interactive Techniques*, 2022.

Zhao, S., Jakob, W., and Li, T.-M. Physics-based differentiable rendering: From theory to implementation. In *SIGGRAPH Courses*, 2020.