EDITOR: Sumanta N. Pattanaik, Sumanta.Pattanaik@ucf.edu

Differentiable Visual Computing: Challenges and Opportunities

Tzu-Mao Li, *University of California, San Diego, CA, 92093, USA*

Classical algorithms typically contain domain-specific insights. This makes them often more robust, interpretable, and efficient. On the other hand, deep-learning models must learn domain-specific insight from scratch from a large amount of data using gradient-based optimization techniques. To have the best of both worlds, we should make classical visual computing algorithms differentiable to enable gradient-based optimization. Computing derivatives of classical visual computing algorithms is challenging: there can be discontinuities, and the computation pattern is often irregular compared to high-arithmetic intensity neural networks. In this article, we discuss the benefits and challenges of combining classical visual computing algorithms and modern data-driven methods, with particular emphasis to my thesis, which took one of the first steps toward addressing these challenges.

Processing and generating visual data, such as images and 3-D content is crucial for applications ranging from autonomous driving, robotics, to photography, virtual reality, and visual effects. Such visual computing tasks involve multiple challenges. We need to model the physical process (e.g., light transport or dynamics). Meanwhile, most visual computing problems are ill-posed (e.g., reconstructing a 3-D scene from a photograph), and cannot be modeled by physics solely. Finally, we need to ensure the implementations of visual computing programs are consistent with the mathematical derivation, while processing millions of pixels and billions of polygons and voxels inside an inference loop.

Modern deep learning has achieved tremendous success, thanks to the power of gradient-based optimization methods to solve nonlinear objectives over many unknowns. However, deep neural networks alone are not sufficient to address the challenges in visual computing. They usually do not model the physical process directly, and have to learn only from data. They require significant computational resources for both training and inference. They are difficult to debug and control: when the network produces undesirable results, it is often difficult to correct for.

On the other hand, classical, domain-specific computer graphics methods suffer less from these issues: they directly model the formation of visual data (e.g., how images are rendered from 3-D scenes, how image edges are related to color differences), and they are usually more interpretable and, thus, are easier to debug, control, and optimize for the performance. However, they often do not apply as broadly as modern data-driven methods, since they do not learn from a large amount of data.

My research, *differentiable visual computing*, aims to connect classical graphics algorithms with modern data-driven methods. A key is to make classical algorithms differentiable to enable gradient-based optimization. Derivatives are useful for both data-driven and nondata-driven scenarios, and for both inverse problems and forward computation (see Figure 1).

Differentiating classical graphics algorithms is challenging. Unlike neural networks, these algorithms often contain irregular computational patterns, which can lead to discontinuities, stochasticity, and complex memory access. For example, rendering algorithms need to handle occlusion and scattering of lights; physics simulation algorithms need to track the movements of quantities stored in different data structures

0272-1716 © 2022 IEEE
Digital Object Identifier 10.1109/MCG.2022.3149550
Date of current version 11 April 2022.

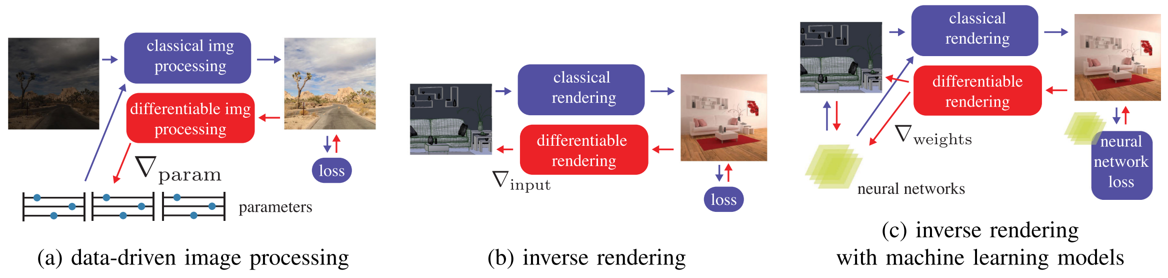


FIGURE 1. Example use cases of differentiable visual computing. (a) Enhancing classical image processing algorithms (edge-aware filtering, tone mapping, etc.) by adding parameters and optimizing them using backpropagation. (b) Searching for the 3-D scene that renders to the observed photograph by backpropagating through rendering. (c) Combining inverse rendering with neural networks by backpropagating through rendering to update the network weights.

such as particles, grids, or meshes; and image processing algorithms need to handle spatially varying kernels and nonlinear resampling.

Traditional automatic differentiation,[17] which is the basis of the backpropagation algorithm in deep-learning methods, does not handle discontinuities caused by occlusion and contact, and does not easily support massive parallelism for the derivative code to efficiently run on modern hardware. Furthermore, automatic differentiation does not tell us how we should use the derivatives. We need algorithms for differentiating graphics programs, and systems for compiling and optimizing the derivative code. The differentiation enables new applications that combine classical methods with data-driven approaches.

In my thesis, with the collaboration of many brilliant researchers, we derived derivatives for rendering while correctly taking discontinuities into account.[24] We also developed a system for differentiating image processing and array programs, and explored applications in combining classical image processing with modern data-driven methods.[25] Since the publication of my thesis, several research groups, including ours, have made advances in this emerging field. Examples include more robust sampling for differentiable rendering,[6,29,48] systems for differentiable physics simulation,[20,21] applications in vector graphics,[26] and a differentiable programming language for parametric discontinuities. [5]

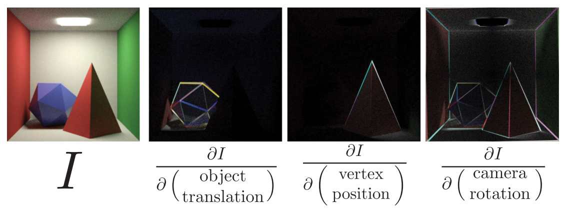

I
$\frac{\partial I}{\partial \left(\text{object translation}\right)}$
$\frac{\partial I}{\partial \left(\text{vertex position}\right)}$
$\frac{\partial I}{\partial \left(\text{camera rotation}\right)}$

FIGURE 2. Differentiable rendering computes the derivatives of images with respect to scene parameters. For each pixel, we need to backpropagate the derivatives with respect to scene parameters, such as object translation, vertex positions, camera poses, or material and light parameters. The derivatives of geometry parameters often concentrate at the object or shadow boundaries.

DIFFERENTIABLE RENDERING FROM FIRST PRINCIPLES

Rendering synthesizes images from a given 3-D scene with geometry, materials, lights, and camera position. By contrast, *differentiable* rendering computes the gradient of a scalar loss of the synthesized image, with respect to the scene parameters, such as camera pose, light intensity, and vertex positions of the triangle mesh (see Figure 2). Having the gradient allows us to use a renderer inside computer vision pipelines: given photographs, we can use gradient descent to find the scene parameters that render to the observed images [see Figure 1(b)]. We can also train deep neural networks with a *rendering loss:* consider a network that outputs a 3-D scene given an image, we want to define a loss function based on the distance between the rendered image and the input image—backpropagating the rendering loss to the network parameters require differentiable rendering [see Figure 1(c)].

One of the key challenges of differentiable rendering is the discontinuities caused by occlusion and object boundaries. Previous work tackled this by approximating the image formation models or the gradients (e.g., Loper and Black's work[28]). Instead, we derive the correct derivatives from the first principles.

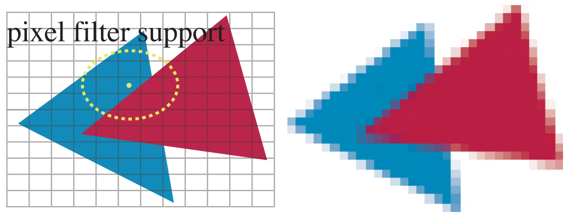


FIGURE 3. To reconstruct the continuous scene from the discrete pixel samples, the color of a pixel is reconstructed by integrating over a *pixel filter*. Rendering is fundamentally differentiable due to this antialiasing process, since antialiased color changes smoothly as the objects move.

We observe that, no matter rastered or ray traced, local shaded or global illuminated, from the signal processing perspective, a pixel is a point sample associated with a reconstruction filter for reconstructing the underlying 2-D signal[42] (see Figure 3). Mathematically, for each pixel color I_x at 2-D position $\mathbf{x}$ in the image space, we are solving an antialiasing integral

$$I_x = \iint_D k(\mathbf{x}')f(\mathbf{x}+\mathbf{x}';p)d\mathbf{x}' \tag{1}$$

where k is the reconstriction filter kernel, and $f(\mathbf{x};p)$ is the color of the 3-D scene at location $\mathbf{x}$, with scene parameter p, and D is the pixel filter support. This corresponds to how human eyes and cameras capture colors: the photoreceptors have finite area; thus, they are integrating photons over the area with a filtering kernel.

Our key idea is based on the fact that, while the scene color f is not differentiable with respect to the parameter p due to object boundaries and occlusion, the pixel color I is differentiable with respect to p after reconstruction. Imagine the rendering of a constant color triangle, the reconstructed pixel color changes smoothly as the triangle moves around (see Figure 3).

However, naive differentiation would not compute the derivative dI/dp correctly. Since the antialiasing integral [see (1)] usually does not have a closed-form solution, we have to evaluate the integral numerically by sampling (rasterization can be seen as sampling at the center). Unfortunately, naive area sampling of the derivative integral fails to account for the discontinuities. Consider a constant-color triangle moving inside a pixel filter (see Figure 4), all of the color changes come from the boundary of the triangle, where area sampling has zero probability to hit. Mathematically, when we swap the derivative operator d/dp inside the integral, the derivative of f contains Dirac delta signals due to the discontinuities, and area sampling misses these Dirac deltas. Our solution is to explicitly sample (Monte Carlo or quadrature), the derivative integral at the discontinuities (or *edges*), in addition to the area sampling (see Figure 4). This leads us to a *consistent* and unbiased estimator for the value of the derivative integral, as we increase the number of samples, the estimation converges to the true integral. Our solution naturally generalizes to soft shadow, global illumination, and other rendering phenomenon that involves integrals. Crucially, our insight provides a principled way to derive differentiable rendering algorithms, even for primary visibility with local shading.

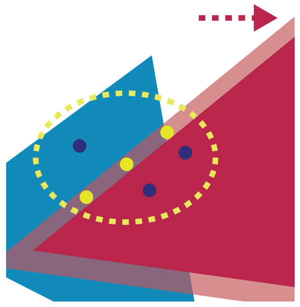


FIGURE 4. When the geometry moves within the pixel filter, the boundary movement is a major source of the change of the integral. However, traditional area sampling (blue samples) does not handle the boundary contribution, since area sampling has zero probability hitting the infinitesimal edges. We propose to explicitly sample the edges (yellow samples) in order to correctly account for the boundary derivatives.

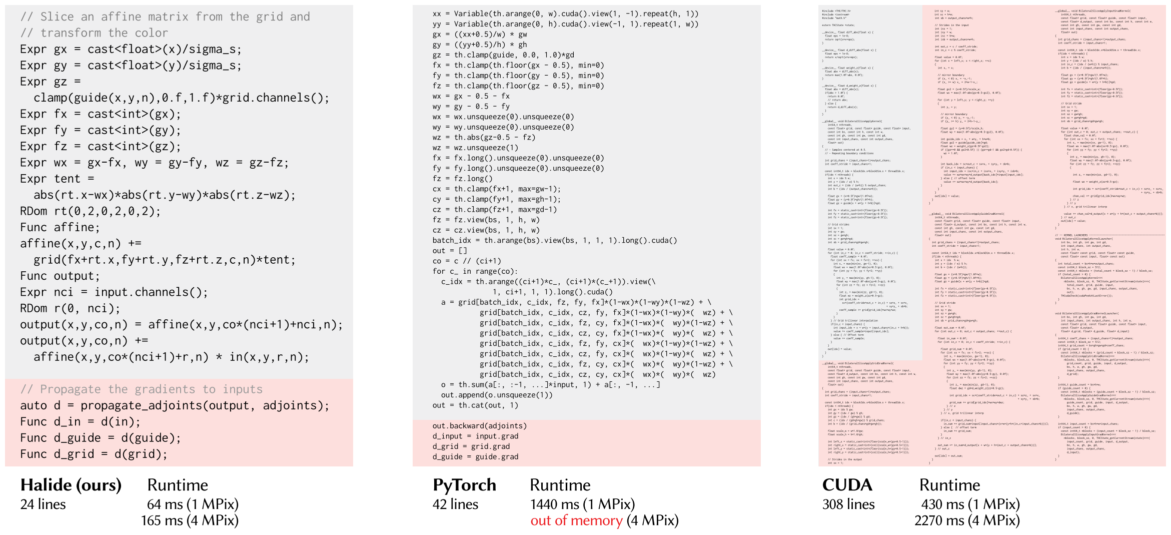
We implemented this idea in a GPU differentiable renderer *redner*.[a] Visit the supplementary website to see some inverse rendering in action.[b] Later in this article, we discuss methods and applications both from our own teams and from other research groups that are built on these ideas.

DIFFERENTIABLE IMAGE PROCESSING WITH HALIDE

Deep-learning frameworks, such as PyTorch[36] or TensorFlow,[1] have made programming deep neural networks so accessible that anyone can program a neural network classifier in a few minutes. The key to the success of deep-learning frameworks is its ability to automatically differentiate neural network architectures, which enables fast iterations of ideas.

These frameworks, however, are designed with neural network *layers* in mind. Neural networks are

[a]https://github.com/BachiLi/redner
[b]https://cseweb.ucsd.edu/~tzli/diffrt/supplementary_webpage/



```
// Slice an affine matrix from the grid and
// transform the color
Expr gx = cast<float>(x)/sigma_s;
Expr gy = cast<float>(y)/sigma_s;
Expr gz =
  clamp(guide(x,y,n),0.f,1.f)*grid.channels();
Expr fx = cast<int>(gx);
Expr fy = cast<int>(gy);
Expr fz = cast<int>(gz);
Expr wx = gx-fx, wy = gy-fy, wz = gz-fz;
Expr tent =
  abs(rt.x-wx)*abs(rt.y-wy)*abs(rt.z-wz);
RDom rt(0,2,0,2,0,2);
Func affine;
affine(x,y,c,n) +=
  grid(fx+rt.x,fy+rt.y,fz+rt.z,c,n)*tent;
Func output;
Expr nci = input.channels();
RDom r(0, nci);
output(x,y,co,n) = affine(x,y,co*(nci+1)+nci,n);
output(x,y,co,n) +=
  affine(x,y,co*(nci+1)+r,n) * in(x,y,r,n);

// Propagate the gradients to inputs
auto d = propagate_adjoints(output, adjoints);
Func d_in = d(in);
Func d_guide = d(guide);
Func d_grid = d(grid);
```

Halide (ours) 24 lines
Runtime
64 ms (1 MPix)
165 ms (4 MPix)

```
xx = Variable(th.arange(0, w).cuda().view(1, -1).repeat(h, 1))
yy = Variable(th.arange(0, h).cuda().view(-1, 1).repeat(1, w))
gx = ((xx+0.5)/w) * gw
gy = ((yy+0.5)/h) * gh
gz = th.clamp(guide, 0.0, 1.0)*gd
fx = th.clamp(th.floor(gx - 0.5), min=0)
fy = th.clamp(th.floor(gy - 0.5), min=0)
fz = th.clamp(th.floor(gz - 0.5), min=0)
wx = gx - 0.5 - fx
wy = gy - 0.5 - fy
wx = wx.unsqueeze(0).unsqueeze(0)
wy = wy.unsqueeze(0).unsqueeze(0)
wz = th.abs(gz-0.5 - fz)
wz = wz.unsqueeze(1)
fx = fx.long().unsqueeze(0).unsqueeze(0)
fy = fy.long().unsqueeze(0).unsqueeze(0)
fz = fz.long()
cx = th.clamp(fx+1, max=gw-1);
cy = th.clamp(fy+1, max=gh-1);
cz = th.clamp(fz+1, max=gd-1)
fz = fz.view(bs, 1, h, w)
cz = cz.view(bs, 1, h, w)
batch_idx = th.arange(bs).view(bs, 1, 1, 1).long().cuda()
out = []
co = c // (ci+1)
for c_ in range(co):
  c_idx = th.arange((ci+1)*c_, (ci+1)*(c_+1)).view(\
            1, ci+1, 1, 1).long().cuda()
  a = grid[batch_idx, c_idx, fz, fy, fx]*(1-wx)*(1-wy)*(1-wz) + \
          grid[batch_idx, c_idx, cz, fy, fx]*(1-wx)*(1-wy)*(  wz) + \
          grid[batch_idx, c_idx, fz, cy, fx]*(1-wx)*(  wy)*(1-wz) + \
          grid[batch_idx, c_idx, cz, cy, fx]*(1-wx)*(  wy)*(  wz) + \
          grid[batch_idx, c_idx, fz, fy, cx]*(  wx)*(1-wy)*(1-wz) + \
          grid[batch_idx, c_idx, cz, fy, cx]*(  wx)*(1-wy)*(  wz) + \
          grid[batch_idx, c_idx, fz, cy, cx]*(  wx)*(  wy)*(1-wz) + \
          grid[batch_idx, c_idx, cz, cy, cx]*(  wx)*(  wy)*(  wz)
  o = th.sum(a[:, :-1, ...]*input, 1) + a[:, -1, ...]
  out.append(o.unsqueeze(1))
out = th.cat(out, 1)

out.backward(adjoints)
d_input = input.grad
d_grid = grid.grad
d_guide = guide.grad
```

PyTorch 42 lines
Runtime
1440 ms (1 MPix)
out of memory (4 MPix)

```
[illegible]
```

CUDA 308 lines
Runtime
430 ms (1 MPix)
2270 ms (4 MPix)

FIGURE 5. Implementations of the forward (gray) and gradient (red) computations of the bilateral slicing layer[15] in Halide, PyTorch, and CUDA. Using our automatic differentiation and scheduling extensions, the Halide implementation is clear, concise, and fast.

usually composed of a few types of regular components from a commonly used toolbox (e.g., convolution and fully connected layers). This, plus the high arithmetic intensity of these layers (they are able to saturate the compute resources of modern computer architectures such as GPUs on their own), has led to a design principle where performance engineers develop high-performance implementation of individual layers, and the users would compose these layers to build their neural networks.

As we move toward general differentiable programming, the design of deep-learning frameworks stops to scale, since we may not be able to efficiently implement our model using common neural network layers. When a user wants to implement a more unconventional network architecture, they often have to implement their *custom layers* in C++ or CUDA. For example, the bilateral slicing layer[15] (see Figure 5) and the more recent KiloNeRF[40] both need to implement custom CUDA code for their unconventional architectures. This leads to a phenomenon where people tend to stick with architectures where existing frameworks are good at, and are unwilling to explore unconventional ones.[7]

My research explores domain-specific languages that allows us to write concise and high-performance code that can be automatically differentiated. We focused on dense array computation—commonly appears in image processing and deep learning. We build on an existing domain-specific language Halide[38] for array processing. Halide's main idea is to separate the high-level algorithm from the low-level *scheduling* (parallelism, order of computation, memory allocation, mapping to GPU blocks and threads, executing on DSPs, etc). For example, to compute the sum of an array, the high-level algorithm would be the summation (e.g., $Y = \sum_{i=1}^{N} X[i]$), and the low-level schedule could be a serial summation running on a CPU, or a parallel hierarchical reduction on a CUDA GPU. Changing the low-level implementation would not change the result of the program, only to make it faster or slower. This separation frees the users from worrying about low-level optimizations while developing the high-level algorithm. They can then explore optimization strategies without unintentionally altering the output.

We extended Halide to automatically and efficiently compute the gradients. Automatic differentiation in Halide enables the differentiation of the high-level algorithms that is agnostic to the low-level implementation, leading to more efficient code. For example, the derivative of the summation is a simple assignment $(dLdX[i] = dLdY)$. However, applying traditional automatic differentiation to the low-level implementation of a parallel hierarchical reduction will likely not preserve the parallelism. In fact, there is no known algorithm currently that can differentiate general imperative parallel programs (say, CUDA code) with the guarantee of preserving the parallelism. By contrast, our method directly differentiates the high-level algorithms in Halide, leading to much simpler code. We can then either ask the user to assign an implementation of the differentiated code, or we can rely on an *autoscheduler*[2,3] to automatically find the

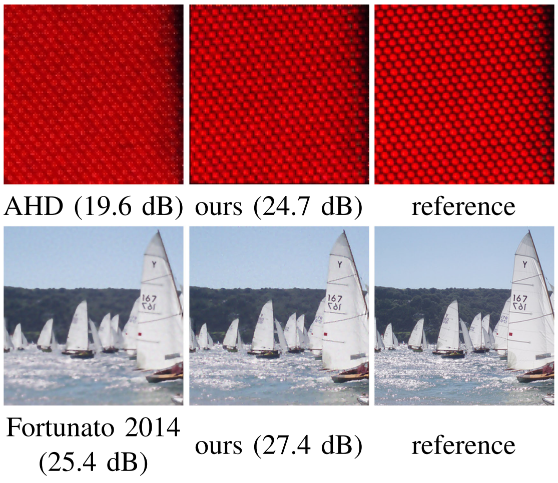


FIGURE 6. We have improved classical image processing algorithms by modifying them with more parameters and optimize them. We modified a demosaicking algorithm AHD[19] by learning a set of 2-D filters, and a nonblind deconvolution algorithm from Fortunato *et al.*[12] by adding multiple stages and learned the parameters.

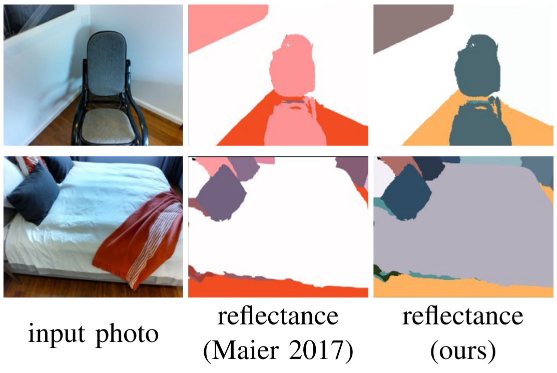


FIGURE 7. Comparison of our differential rendering enabled material reconstruction[4] with a traditional computer vision method.[31] We obtain higher fidelity reconstruction due to the ability to account for interreflections and shadows.

best implementation. In this article,[25] we further discuss strategies to further improve parallelism through scatter-gather transformation. This is all possible thanks to Halide's domain-specific separation of high-level algorithm and low-level implementation.

With our system, we are able to concisely and efficiently implement unconventional neural network layers that are difficult to implement in deep-learning frameworks (see Figure 5).

Our vision is that any image-processing pipelines can benefit from an automatic tuning of internal parameters. This step is traditionally done by hand through user trial-and-error. The availability of automatic derivatives makes it possible to systematically optimize any internal parameter of an image processing pipeline, given some output objectives. We show how to significantly improve the performance of two traditional image processing algorithms by slightly modifying the method to introduce more parameters and automatically optimize them (see Figure 6).

IMPACTS AND FOLLOWUPS

Applications of Differentiable Rendering

Differentiable rendering opens up a wide range of applications. While "vision as inverse graphics" is not a new idea,[8] our work, along with other work in vision and graphics (e.g., Gkioulekas *et al.*'s,[16] Loper and Black's,[28] Kato *et al.*'s,[22] and Liu *et al.*'s[27] work) revitalized a trend of unifying 3-D reconstruction by bringing rendering in the loop. Differentiable rendering can be used with classical computer vision, such as multiview or photometric stereo, by using the classical methods as an initial guess, then refine the results using gradient descent on a loss function that incorporates rendering. Our recent work[4] reconstructs lighting and materials from images (see Figure 7) for augmented reality applications. Nimier-David *et al.*[34] later improved the method to handle high-resolution textures. Luan *et al.*[30] and Dib *et al.*[10] used our differentiable rendering algorithm to reconstruct the geometry and materials from images for general 3-D objects and faces, respectively. Compared to classical vision, these methods can often obtain higher fidelity results thanks to the renderer in the loop.

As discussed earlier, differentiable rendering can be used for training machine learning models to output 3-D scenes from input images [see Figure 1(c)]. Using our algorithm, Dib *et al.*[11] trained a network to predict the geometry and reflectance of faces from an image, while Griffiths *et al.*'s network predicted the scene structures.[18] Using their custom renderer, Che *et al.*[9] trained an autoencoder for recovering scattering coefficients of participating media.

Differentiable rendering is also useful outside of the 3-D rendering domain. We demonstrated applications for editing and learning vector graphics (see Figure 8), through differentiating vector graphics rasterization using the same principle as the edge sampling.[26] Our work has led to many followups on learning-based vector graphics generation: using our differentiable rasterizer, Reddy *et al.*[39] trained a network to synthesize vector graphics from raster

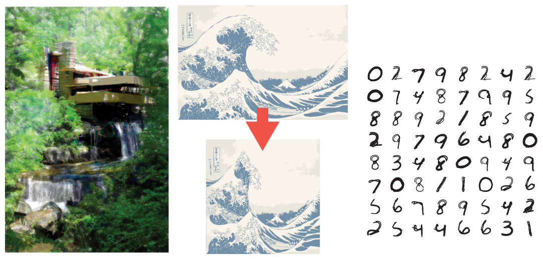


FIGURE 8. Differentiable rendering can be used for learning and editing vector graphics. In a recent work,[26] we showed applications such as painterly rendering, seam carving for vector graphics, and a generative model that synthesize vector graphics using raster training data alone.

training data; Frans *et al.*[13] and Schaldenbrand *et al.*[41] combined our differentiable rasterizer with CLIP,[37] a neural network that measures the distance between text and images, to synthesize vector graphics from text.

Faster and More General Differentiable Rendering

Making rendering fast is hard. We need to resolve occlusion between object pairs, stream the data to maximize locality, and evaluate integrals numerically with minimal errors. While modern rendering engines are blazingly fast for certain inputs, challenges still remain for making forward rendering robust and fast for arbitrary 3-D scenes.

Differentiable rendering introduces extra performance challenges, the discontinuity derivatives requires different evaluation strategies compared to the original rendering process, and the derivative computation introduces different memory traffic that may need different performance optimization or even different hardware.

Currently research focuses on making differentiable rendering computation as similar to existing forward rendering pipelines as possible. Laine *et al.*[23] made differentiable rendering compatible with existing rasterization hardware, by adopting the insight from our method at the highest level: antialiased rendering is differentiable. They designed a deferred shading pipeline with a postprocessing-based antialiasing, which allows them to avoid the discontinuity sampling that requires ray tracing operation.

Much other work focused on the ray tracing regime. Loubet *et al.*[29] proposed to use area sampling (instead of the discontinuity sampling) of a ray sample's neighborhood to estimate a local reparametrization of the integral to remove the discontinuity. Their method allows us to use traditional acceleration structures for differentiable rendering. Our followup work with Bangaru[6] built on their method, and showed that their reparametrization is equivalent to applying divergence theorem to transform the integral over the discontinuities to over the whole area. We then derived the correct criteria for consistent and unbiased derivatives estimation. On the other hand, Zhang *et al.*[50] discovered that the discontinuity sampling is easier if it is done in the *path-space*. This means that we can sample points on the geometric boundaries, then connect those points to the camera and light sources.

Even when not considering discontinuities, differentiable rendering introduces different tradeoffs in memory, compute, and sampling efficiency. Zeltner *et al.*[46] and Zhang *et al.*[47] discussed importance sampling. Nimier-David *et al.*[35] and Vicini *et al.*[43] discussed memory and locality optimization.

Finally, recent work has generalized differentiable rendering for handling participating media,[49,50] time-gated rendering,[44] and transient rendering.[45]

Differentiable rendering versus neural rendering

Recent work in vision and graphics have shown that neural networks can serve as powerful 3-D representations and can be used for complementing or replacing traditional data structures, such as meshes, grids, textures, and low-dimensional parametric functions. Mathematically, the abovementioned differentiable rendering methods are compatible with neural network representations—rendering equation assumes nothing about the scene representation. Computation-wise, making these algorithms efficient for neural network representations requires more efforts and is an exciting research avenue. For example, neural radiance fields (NeRF)[33] use a network to represent an emissive volume with spatio-directionally varying absorption and emission coefficients—a special case of the standard radiative transfer equation. Generalizing NeRF to handle multiple-scattering and combining it with other surface primitives, while making both the forward and differentiation computation efficient, requires extensive performance engineering and likely algorithmic innovations.

Domain-Specific Languages for Differentiable Visual Computing

Our differentiable image processing work with Halide reveals a new design space for image processing

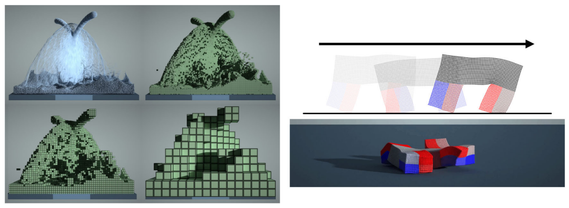
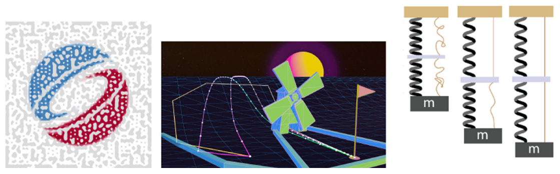


FIGURE 9. Traditionally, implementing hierarchical sparse data structures for physical simulation efficiently is challenging and requires tedious and lengthy programs. Our compiler Taichi[21] automates the implementation and allows programmers to employ hierarchical sparse data structures in their physical simulation code with minimal efforts (left). Taichi also supports automatic differentiation,[20] which enables applications such as training the neural network controllers of a soft body robot (right).



(a) inverse shader design (b) trajectory optimization (c) stress-strain optimization

FIGURE 10. Applications of our Teg prototype programming language[5] that can differentiate discontinuities and integrate the resulting Dirac delta. (a) Optimizing parameters of a Perlin noise shader. (b) Optimizing a physical trajectory with contact. (c) Optimizing the parameter of a strain-limiting mass spring model with discontinuous spring constants.

pipelines, between the bulky and overparametrized convolutional neural networks and light-weight hand-designed algorithms that include domain-specific knowledge. For example, the state-of-the-art deep-learning-based demosaicking algorithm[14] delivers impressive reconstruction quality using a fairly conventional convolutional neural networks (CNNs), but it requires over 130,000 operations per pixel, taking teraflops to process a single image. The best classical methods only require hundreds of operations per pixel, and are orders of magnitude more efficient in practice, so they remain dominant in real-world systems in spite of their lesser quality. There is a huge space of the Pareto frontier of the performance and image quality left unexplored. This gap can be bridged by taking the structure of classical image processing algorithms, and learn the parameters of the resulting differentiable program.

As mentioned, these lightweight, low arithmetic intensity image processing pipelines require the fundamentally different performance optimization compared to high arithmetic intensity neural networks. While Halide[25,38] allows the separation between high-level algorithm and low-level scheduling, users still need to find the best low-level optimization themselves. We are working on automatic methods to search for the best schedule given an algorithm.[2,3] The main challenge is that there are typically millions or billions of possible schedules for a given algorithm, making brute force benchmarking infeasible. Our idea is to combine classical static program analysis with a learning-based cost model to produce a lightweight performance predictor, for both selecting and enumerating the best schedule. This idea has also led to advances in deep-learning compilers[51] and traditional compilers.[32]

Our work on differentiable programming motivated us to explore other domain-specific systems. Inspired by Halide's idea of decoupling high-level algorithm and low-level schedule, our work Taichi[21] decouples the data structure design from computation. In particular, Taichi focuses on hierarchical sparse grids—a static tree data structure for representing spatially coherent sparse data (see Figure 9, left), which is crucial for soft bodies and fluids simulation. In Taichi, users access sparse arrays as if they are dense, and specify the static data structures hierarchy separately. The compiler then takes the array access code and the specified sparse data structure, and automatically synthesizes vectorized code for maintaining the data structure and reading/writing the values. We later extended Taichi with automatic differentiation,[20] and showed optimal control/model-based reinforcement learning applications that involve differentiating through physics simulation (see Figure 9, right).

We have recently started to expand the scope of automatic differentiation. As shown in our differentiable rendering work, traditional automatic differentiation ignores the Dirac delta signals that occur when differentiating control flows, such as if/else conditions. This is because the automatic differentiation compiler does not have the semantics of *integrals* to integrate over the Dirac delta signals. Our recent work, Teg[5] explores a programming language for differentiating discontinuities, by including an integral primitive. Teg allows us to generalize our ideas in differentiable rendering to a broader domain, and enables applications including inverse shader design, trajectory optimization, and designing physical objects (see Figure 10). Teg only scratches the surface of automatically differentiating discontinuities and presents an exciting research avenue: supporting modular programming, decoupling importance and stratified sampling, and handling differential equations are all exciting future work.

CONCLUSION

Our vision is that, in the future, there is no clear distinction between deep learning and classical methods. Deep-learning architectures should become more domain-specific to adapt to the problems, while classical methods should become data-driven and adapt to the data. To achieve this, we need new algorithms that are blends between classical methods and deep learning, and we need new systems and domain-specific programming languages that enable fast prototyping of ideas.

REFERENCES

1. M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," *Operating Syst. Des. Implementation*, pp. 265–283, 2016.
2. A. Adams *et al.*, "Learning to optimize Halide with tree search and random programs," *ACM Trans. Graphics*, vol. 38, no. 4, 2019, Art. no. 121.
3. L. Anderson, A. Adams, K. Ma, T.-M. Li, T. Jin, and J. Ragan-Kelley, "Efficient automatic scheduling of imaging and vision pipelines for the GPU," *Proc. ACM Program. Lang.*, vol. 5, 2021, Art. no. 109.
4. D. Azinović, T.-M. Li, A. Kaplanyan, and M. Nießner, "Inverse path tracing for joint material and lighting estimation," in *Proc. IEEE/CVF Comput. Vis. Pattern Recognit.*, 2019, pp. 2442–2451.
5. S. Bangaru, J. Michel, K. Mu, G. Bernstein, T.-M. Li, and J. Ragan-Kelley, "Systematically differentiating parametric discontinuities," *ACM Trans. Graphics*, vol. 40, no. 107, 2021, Art. no. 107.
6. S. P. Bangaru, T.-M. Li, and F. Durand, "Unbiased warped-area sampling for differentiable rendering," *ACM Trans. Graphics*, vol. 39, no. 6, 2020, Art. no. 245.
7. P. Barham and M. Isard, "Machine learning systems are stuck in a rut," in *Proc. Workshop Hot Topics Operating Syst.*, 2019, pp. 177–183.
8. V. Blanz and T. Vetter, "A morphable model for the synthesis of 3D faces," in *Proc. 26th Annu. Conf. Comput. Graphics Interactive Techn.,*1999, pp. 187–194.
9. C. Che, F. Luan, S. Zhao, K. Bala, and I. Gkioulekas, "Towards learning-based inverse subsurface scattering," in *Proc. IEEE Int. Conf. Comput. Photogr.*, 2020, pp. 1–12.
10. A. Dib *et al.*, "Practical face reconstruction via differentiable ray tracing," *Comput. Graphics Forum*, vol. 40, pp. 153–164, 2021.
11. A. Dib, C. Thebault, J. Ahn, P.-H. Gosselin, C. Theobalt, and L. Chevallier, "Towards high fidelity monocular face reconstruction with rich reflectance using self-supervised learning and ray tracing," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 12819–12829.
12. H. E. Fortunato and M. M. Oliveira, "Fast high-quality non-blind deconvolution using sparse adaptive priors," *Vis. Comput.*, vol. 30, no. 6–8, pp. 661–671, 2014.
13. K. Frans, L. Soros, and O. Witkowski, "CLIPDraw: Exploring text-to-drawing synthesis through language-image encoders," 2021, *arXiv:2106.14843*.
14. M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Trans. Graphics*, vol. 35, no. 6, 2016, Art. no. 191.
15. M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, "Deep bilateral learning for real-time image enhancement," *ACM Trans. Graphics*, vol. 36, no. 4, 2017, Art. no. 118.
16. I. Gkioulekas, S. Zhao, K. Bala, T. Zickler, and A. Levin, "Inverse volume rendering with material dictionaries," *ACM Trans. Graphics*, vol. 32, no. 6, Nov. 2013, Art. no. 162.
17. A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed. Philadelphia, PA, USA: Soc. Ind. Appl. Math., 2008.
18. D. Griffiths, J. Boehm, and T. Ritschel, "Curiosity-driven 3D object detection without labels," in *Proc. Int. Conf. 3D Vis.*, 2021, pp. 525–534.
19. K. Hirakawa and T. W. Parks, "Adaptive homogeneity-directed demosaicing algorithm," *IEEE Trans. Image Process.*, vol. 14, no. 3, pp. 360–369, Mar. 2005.
20. Y. Hu *et al.*, "Difftaichi: Differentiable programming for physical simulation," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1eB5xSFvr
21. Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, "Taichi: A language for high-performance computation on spatially sparse data structures," *ACM Trans. Graphics*, vol. 38, no. 6, p. 16, 2019, Art. no. 201.
22. H. Kato, Y. Ushiku, and T. Harada, "Neural 3D mesh renderer," in *Proc. IEEE/CVF Comput. Vis. Pattern Recognit.*, 2018, pp. 3907–3916.
23. S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila, "Modular primitives for high-performance differentiable rendering," *ACM Trans. Graphics*, vol. 39, no. 6, p. 14, 2020, Art. no. 194.
24. T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, "Differentiable Monte Carlo ray tracing through edge sampling," *ACM Trans. Graphics*, vol. 37, no. 6, 2018, Art. no. 222.
25. T.-M. Li, M. Gharbi, A. Adams, F. Durand, and J. Ragan-Kelley, "Differentiable programming for image processing and deep learning in Halide," *ACM Trans. Graphics*, vol. 37, no. 4, 2018, Art. no. 139.
26. T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley, "Differentiable vector graphics rasterization for editing and learning," *ACM Trans. Graphics*, vol. 39, no. 6, 2020, Art. no. 193.

27. S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3D reasoning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 7707–7716.
28. M. M. Loper and M. J. Black, "OpenDR: An approximate differentiable renderer," in *Proc. Eur. Conf. Comput. Vis.*, 2014, vol. 8695, pp. 154–169.
29. G. Loubet, N. Holzschuch, and W. Jakob, "Reparameterizing discontinuous integrands for differentiable rendering," *ACM Trans. Graphics*, vol. 38, no. 6, 2019, Art. no. 228.
30. F. Luan, S. Zhao, K. Bala, and Z. Dong, "Unified shape and SVBRDF recovery using differentiable Monte Carlo rendering," *Comput. Graphics Forum*, vol. 40, no. 4, pp. 101–113, 2021.
31. R. Maier, K. Kim, D. Cremers, J. Kautz, and M. Nießner, "Intrinsic3D: High-quality 3D reconstruction by joint appearance and geometry optimization with spatially-varying lighting," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 3114–3122.
32. C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4505–4515.
33. B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405–421.
34. M. Nimier-David, Z. Dong, W. Jakob, and A. Kaplanyan, "Material and lighting reconstruction for complex indoor scenes with texture-space differentiable rendering," in *Proc. Eurographics Symp. Rendering—DL-only Track*, 2021, pp. 73–84.
35. M. Nimier-David, S. Speierer, B. Ruiz, and W. Jakob, "Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering," *ACM Trans. Graphics*, vol. 39, no. 4, Jul. 2020, Art. no. 146.
36. A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.
37. A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *Proc. Int. Conf. Mach. Learn.*, 2021, vol. 139, pp. 8748–8763.
38. J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling algorithms from schedules for easy optimization of image processing pipelines," *ACM Trans. Graphics*, vol. 31, no. 4, Jul. 2012, Art. no. 32.
39. P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra, "Im2vec: Synthesizing vector graphics without vector supervision," in *Proc. Comput. Vis. Pattern Recognit.*, 2021, pp. 7342–7351.
40. C. Reiser, S. Peng, Y. Liao, and A. Geiger, "KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 14335–14345.
41. P. Schaldenbrand, Z. Liu, and J. Oh, "StyleCLIPDraw: Coupling content and style in text-to-drawing synthesis," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop Mach. Learn. Des.*, 2021.
42. A. R. Smith, "A pixel is not a little square, a pixel is not a little square, a pixel is not a little square!(and a voxel is not a little cube)," *Microsoft Research, Redmond, WA, USA, Tech. Memo 6*, 1995.
43. D. Vicini, S. Speierer, and W. Jakob, "Path replay backpropagation: Differentiating light paths using constant memory and linear time," *ACM Trans. Graphics*, vol. 40, no. 4, 2021, Art. no. 108.
44. L. Wu, G. Cai, R. Ramamoorthi, and S. Zhao, "Differentiable time-gated rendering," *ACM Trans. Graphics*, vol. 40, no. 6, 2021, Art. no. 287.
45. S. Yi, D. Kim, K. Choi, A. Jarabo, D. Gutierrez, and M. H. Kim, "Differentiable transient rendering," *ACM Trans. Graphics*, vol. 40, no. 6, 2021, Art. no. 286.
46. T. Zeltner, S. Speierer, I. Georgiev, and W. Jakob, "Monte Carlo estimators for differential light transport," *ACM Trans. Graphics*, vol. 40, no. 4, 2021, Art. no. 78.
47. C. Zhang, Z. Dong, M. Doggett, and S. Zhao, "Antithetic sampling for Monte Carlo differentiable rendering," *ACM Trans. Graphics*, vol. 40, no. 4, 2021, Art. no. 77.
48. C. Zhang, B. Miller, K. Yan, I. Gkioulekas, and S. Zhao, "Path-space differentiable rendering," *ACM Trans. Graphics*, vol. 39, no. 6, 2020, Art. no. 143.
49. C. Zhang, L. Wu, C. Zheng, I. Gkioulekas, R. Ramamoorthi, and S. Zhao, "A differential theory of radiative transfer," *ACM Trans. Graphics*, vol. 38, no. 6, 2019, Art. no. 227.
50. C. Zhang, Z. Yu, and S. Zhao, "Path-space differentiable rendering of participating media," *ACM Trans. Graphics*, vol. 40, no. 4, 2021, Art. no. 76.
51. L. Zheng *et al.*, "Ansor: Generating high-performance tensor programs for deep learning," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 863–879.

TZU-MAO LI is currently an Assistant Professor with the University of California, San Diego, CA, USA. He worked with Yung-Yu Chuang at the Communication and Multimedia Lab. He received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Taipei City, Taiwan, in 2011 and 2013, respectively, and the Ph.D. degree from the Massachusetts Institute of Technology, Cambridge, MA, USA, advised by Frédo Durand. He was the recipient of the ACM SIGGRAPH 2020 Outstanding Doctoral Dissertation Award.

Contact department editor Sumanta N. Pattanaik at Sumanta.Pattanaik@ucf.edu.