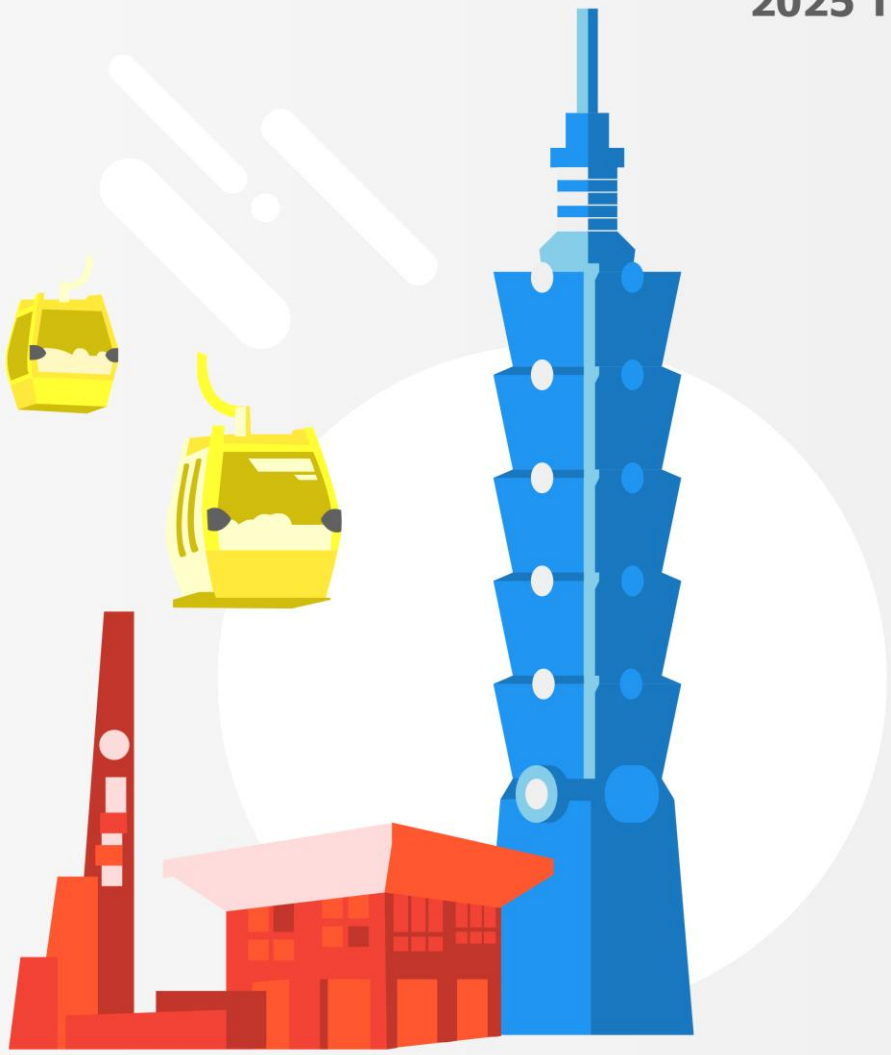
2025 TAIPEI

Geometric Integration for Neural Control Variates

Daniel Meister and Takahiro Harada

AMD



Computer Graphics Problems

Many problems in computer graphics can be formulated as integral equations

- Geometric processing (e.g., Laplace equation)
- Light transport (e.g., the rendering equation)

$$L(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\boldsymbol{\omega}_i \in \Omega} f_r(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) L(\mathbf{x}, \boldsymbol{\omega}_i) \cos\theta_r \, \mathrm{d}\boldsymbol{\omega}_i$$

No analytic solution in general → numerical integration

All problems can be formulated as a general integral on some domain

$$F = \int_{\mathcal{D}} f(x) \, \mathrm{d}x$$

Monte Carlo (MC) Integration

Numerical integration widely used in computer graphics

- High-dimensional problems
- Robust to discontinuities
- Converges to the true solution
  - Many samples needed to surpass the variance (noise)

$$F = \int_{\mathcal{D}} f(x)\,\mathrm{d}x \approx \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{p(x_i)}$$

Variance reduction techniques

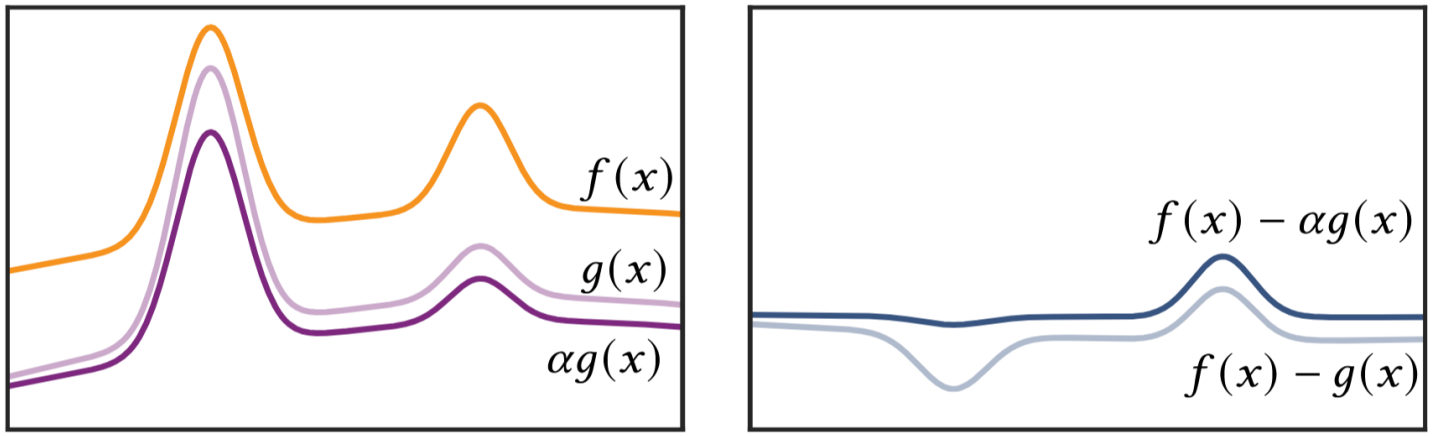
- Importance sampling
- **Control variates**

Monte Carlo with Control Variates

Approximate the integrand and integrate only the residual integral

- We assume $\alpha = 1$ from now on

$$F = \int_{\mathcal{D}} f(x)\,\mathrm{d}x = \alpha G + \int_{\mathcal{D}} f(x) - \alpha g(x)\,\mathrm{d}x \qquad G = \int_{\mathcal{D}} g(x)\,\mathrm{d}x$$



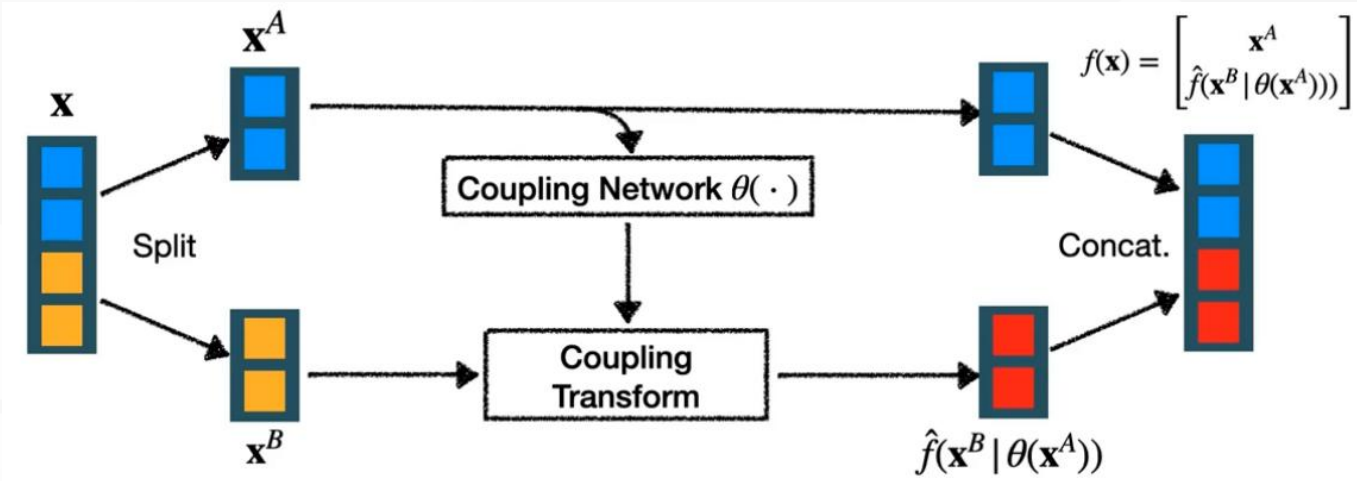
Courtesy of [Muller et al. 2020]

Neural networks are universal approximators. Why not use them as control variates?

Integrating neural network is challenging

Prior Work - Normalizing Flows [Muller et al. 2020]

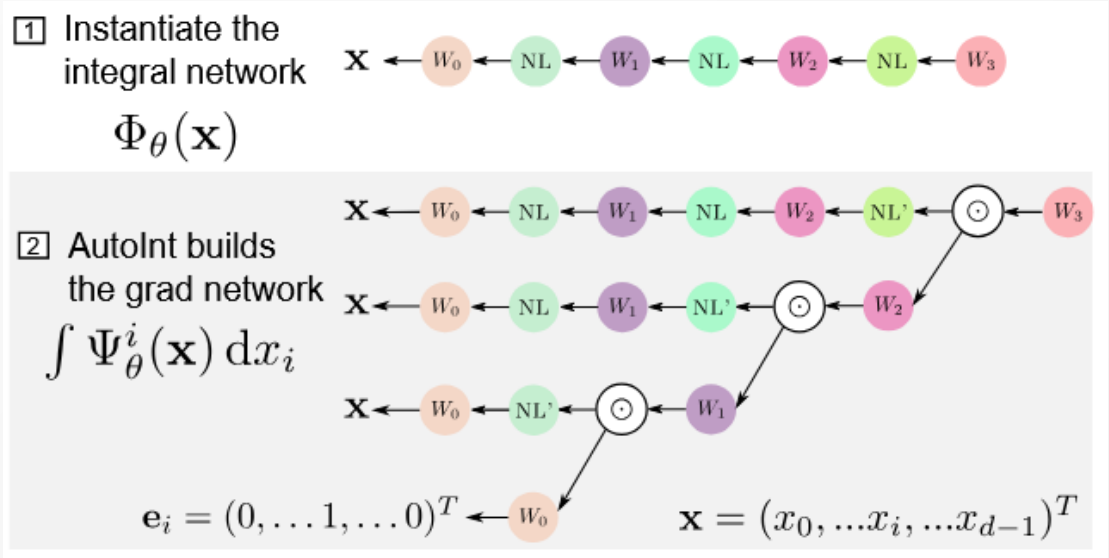
- Normalized probability distribution
  - Scaled by another neural network
  - Scale trivially corresponds to *G*
- A chain of coupling layers (small neural networks)
  - Invertibility limits expressiveness
  - We do not need an inverse mapping
- Does not work well in low dimensions



Courtesy of [Brubaker et al. 2020]

Prior Work – Automatic Integration [Lindell et al. 2021, Li et al. 2024]

- Neural network represents antiderivative
  - A pair of networks: grad and integral networks
  - Grad network is constructed based on the integral network
- Requires autodiff and higher order derivatives
  - The integral network must be differentiable enough
  - Non-zero *n*-th order derivatives for *n*-dimensions
- Grad network can be huge
  - Difficult training
  - Size scales with the number of dimensions



Courtesy of [Lindell et al. 2021]

Contribution

Control variates using the MLP with *piece-wise linear activation functions* (ReLU)

- Analytic integration of the MLP as a geometric problem
- Employing tools from computational geometry to solve it in 2D
- Light transport applications

An MLP with piece-wise linear activation functions represents a piece-wise linear (affine) function

MLP = Multilayer perceptron
ReLU = Rectified linear unit

Geometric Integration

- Split integration domain into disjoint subdomains
- Such that the function is affine on the subdomain
- Integrate the corresponding affine function on each subdomain

$$G = \int_{\mathcal{D}} g(x)\,\mathrm{d}x = \sum_{i=1}^{n} \int_{\mathcal{D}_i} g(x)\,\mathrm{d}x = \sum_{i=1}^{n} \int_{\mathcal{D}_i} g_i(x)\,\mathrm{d}x \qquad \bigcup_{i=1}^{n} \mathcal{D}_i = \mathcal{D} \qquad \bigcap_{i=1}^{n} = \emptyset$$

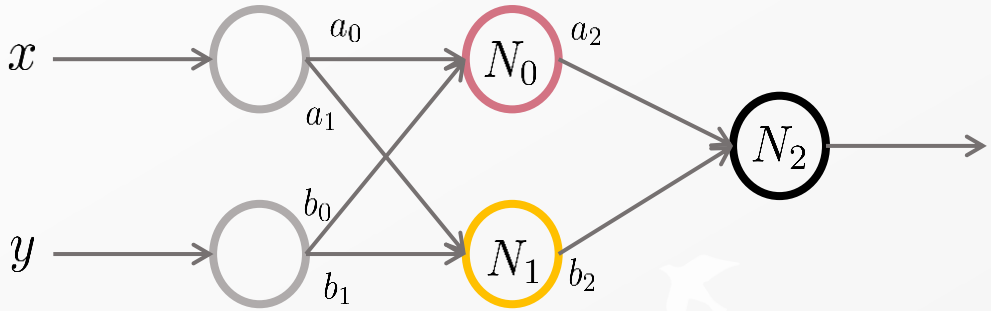
- Subdomains are convex regions
  - Intersection of half-planes
  - Half-planes correspond to inequalities in the activation functions
- The resulting function is always affine
  - Affine functions are closed under composition
  - Composition of linear layers and piece-wise linear activation functions

Finding the subdomains is the challenging part

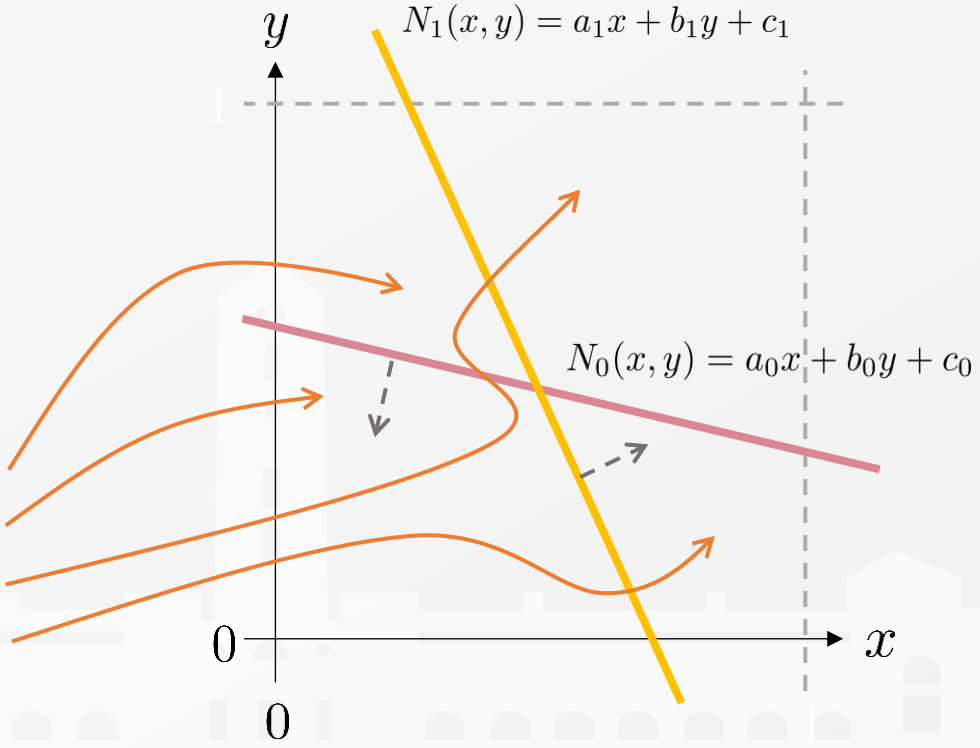
Geometric Integration – Example

MLP with one hidden layer and ReLU activation functions

- Integration domain is a unit square
- Each neuron corresponds to a half-plane (an oriented line)
- The resulting function depends on the neuron activations



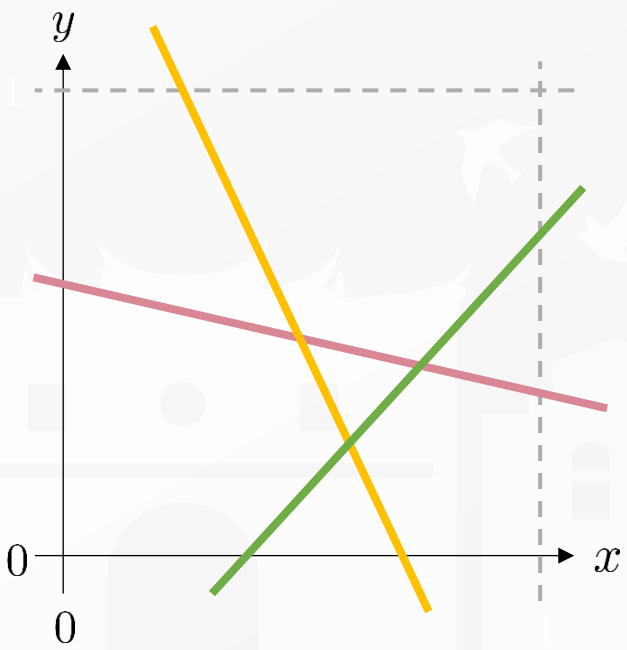
$$N_2(x,y) = \begin{cases} c_2 & \text{if } N_0(x,y) < 0 \text{ and } N_1(x,y) < 0 \\ a_2(a_0x + b_0y + c_0) + c_2 & \text{if } N_0(x,y) \geq 0 \text{ and } N_1(x,y) < 0 \\ b_2(a_1x + b_1y + c_1) + c_2 & \text{if } N_0(x,y) < 0 \text{ and } N_1(x,y) \geq 0 \\ a_2(a_0x + b_0y + c_0) + b_2(a_1x + b_1y + c_1) + c_2 & \text{if } N_0(x,y) \geq 0 \text{ and } N_1(x,y) \geq 0 \end{cases}$$

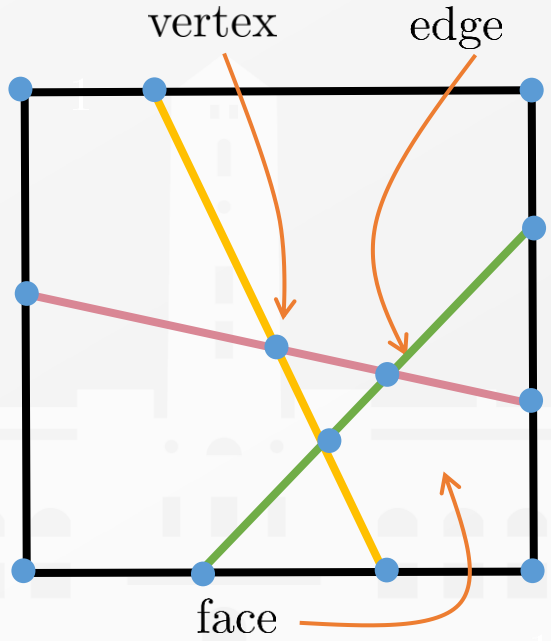


We need to think in terms of affine functions instead of numbers!

Arrangement of Lines

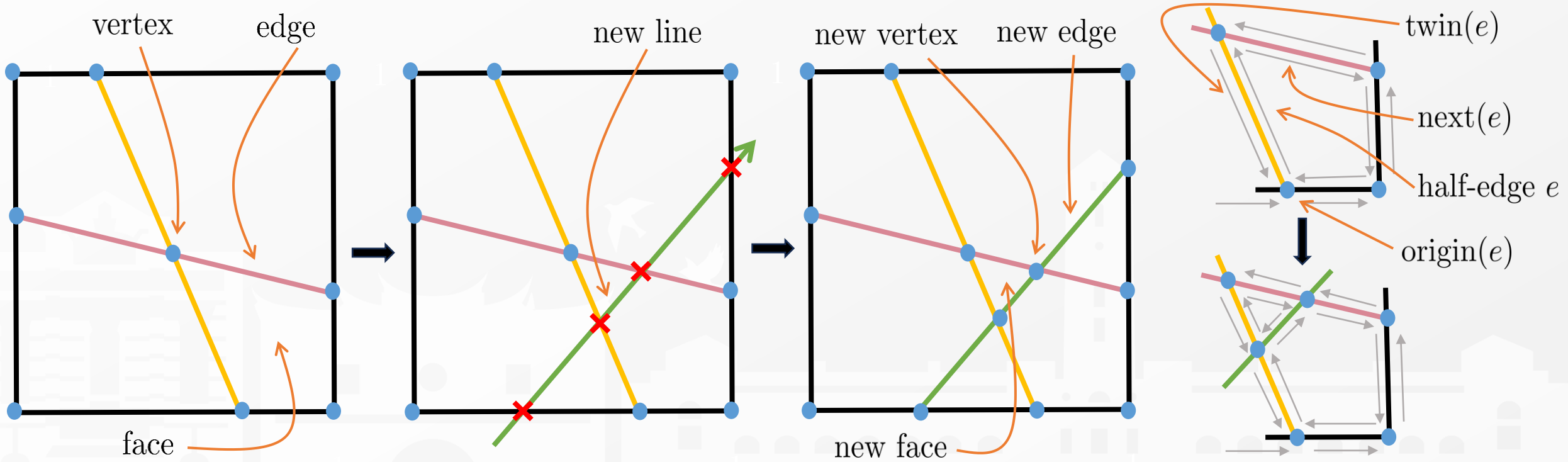
- Well-studied problem in computational geometry
- Find a planar subdivision induced by given a set of lines in 2D
  - Vertices – points where lines meet (no more than $\binom{N}{2}$)
  - Edges – line segments between points (no more than N^2)
  - Faces – convex regions containing no lines (no more than $\binom{N}{2} + N + 1$)
  - N is the number of lines (the number of neurons in the a hidden layer)





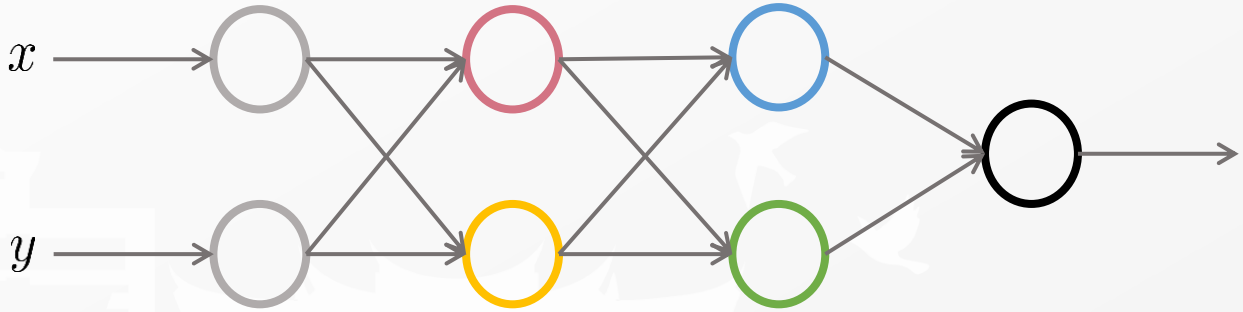
Doubly-Connected Edge List (DCEL)

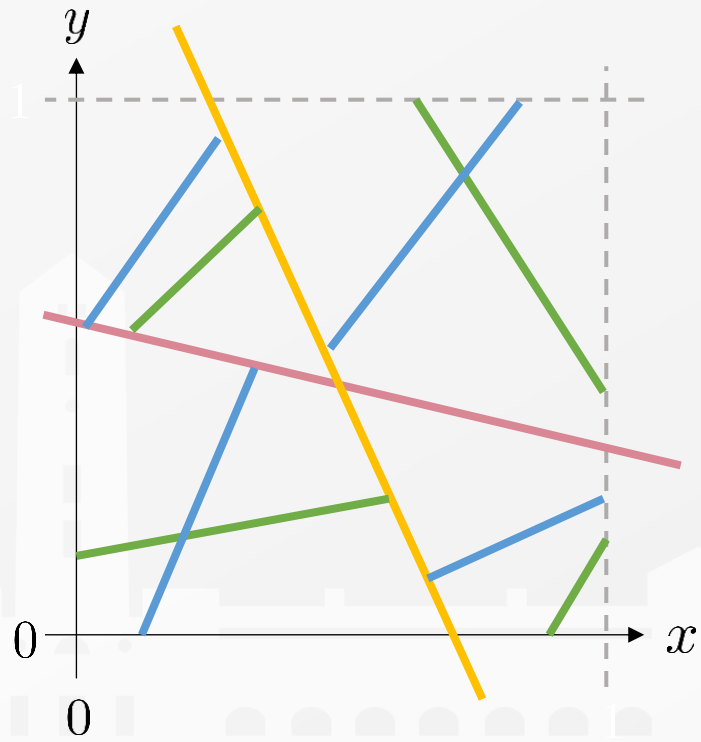
- A data structure for storing a planar subdivision
- Topological information
- Incremental construction – “2D ray tracing”



Multiple Hidden Layers

- The half-planes are mixed based on the activation from the previous layer
- Different activations result in different half-planes in each face
- Recursive processing by traversing “very wide tree”
  - The number of neurons in a layer corresponds to the branching factor





We can relatively easily integrate the corresponding affine function once the subdomain is determined

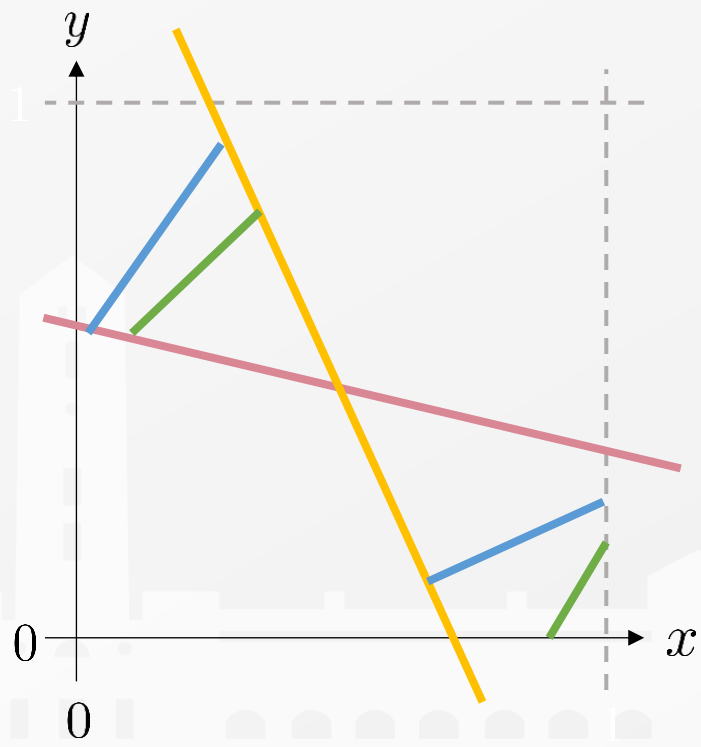
Please, see the details in the paper

Traversal Algorithm

- The goal is to visit all bottom-most “leaf” faces
  - Simple stackless logic
- No need to store the whole subdivision
  - Once the face at any level is processed
  - The allocated DCELs can be reused for other faces
- Allocating only one DCEL per layer

Missing details

- The lines are oriented (half-planes)
- Track the resulting affine function (for the integration)

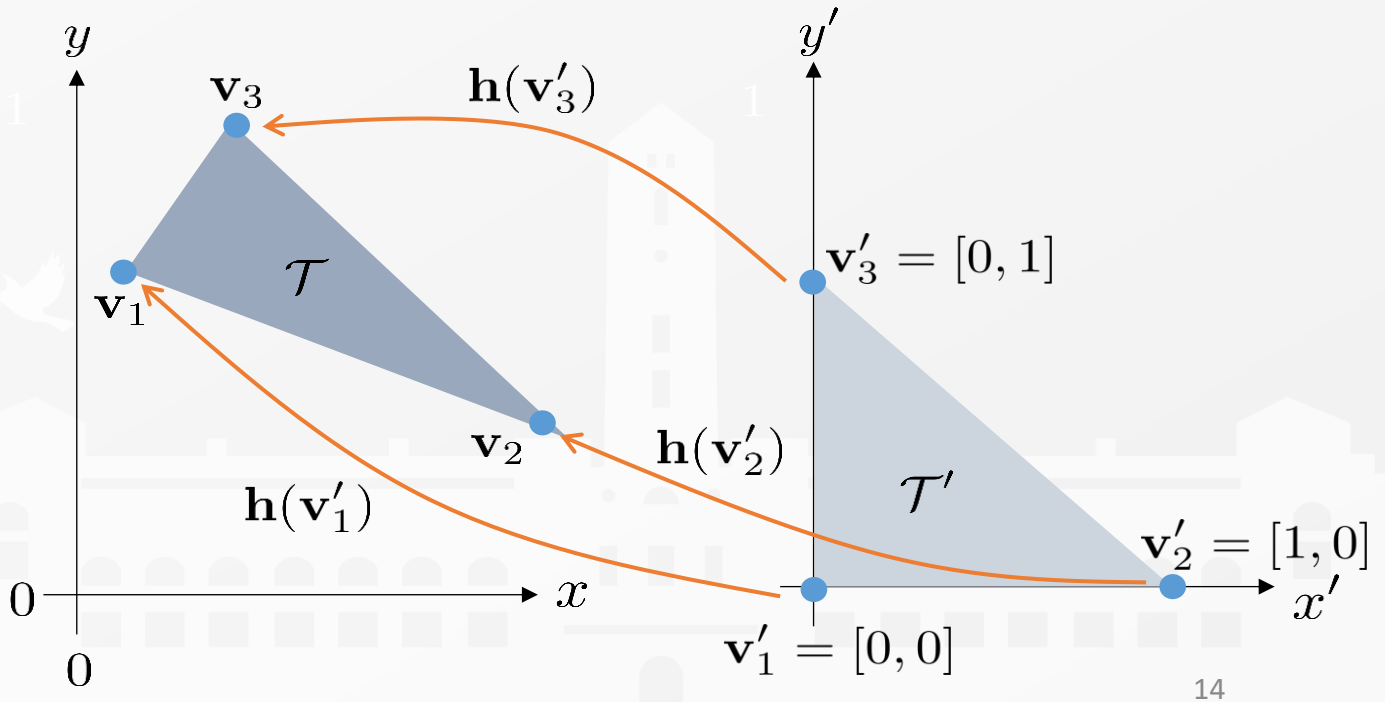


Triangle Integration

- Convex regions can be trivially triangulated
- One of the variable needs to be independent
  - Iterative (nested) integrals computed as 2x 1D integrals
- Integrating over a unit triangle
  - Mapping back to the original one
  - The mapping is an affine function

$$G = \sum_{i=1}^{n} \int_{\mathcal{D}_i} g_i(x, y)\,\mathrm{d}y\,\mathrm{d}x = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \int_{\mathcal{T}_{i,j}} a_i x + b_i y + c_i\,\mathrm{d}y\,\mathrm{d}x$$

$$2A_{\mathcal{T}} \int_0^1 \int_0^{1-x'} a'x' + b'y' + c'\,\mathrm{d}y'\,\mathrm{d}x' = \frac{A_{\mathcal{T}}}{3}(a' + b' + 3c')$$

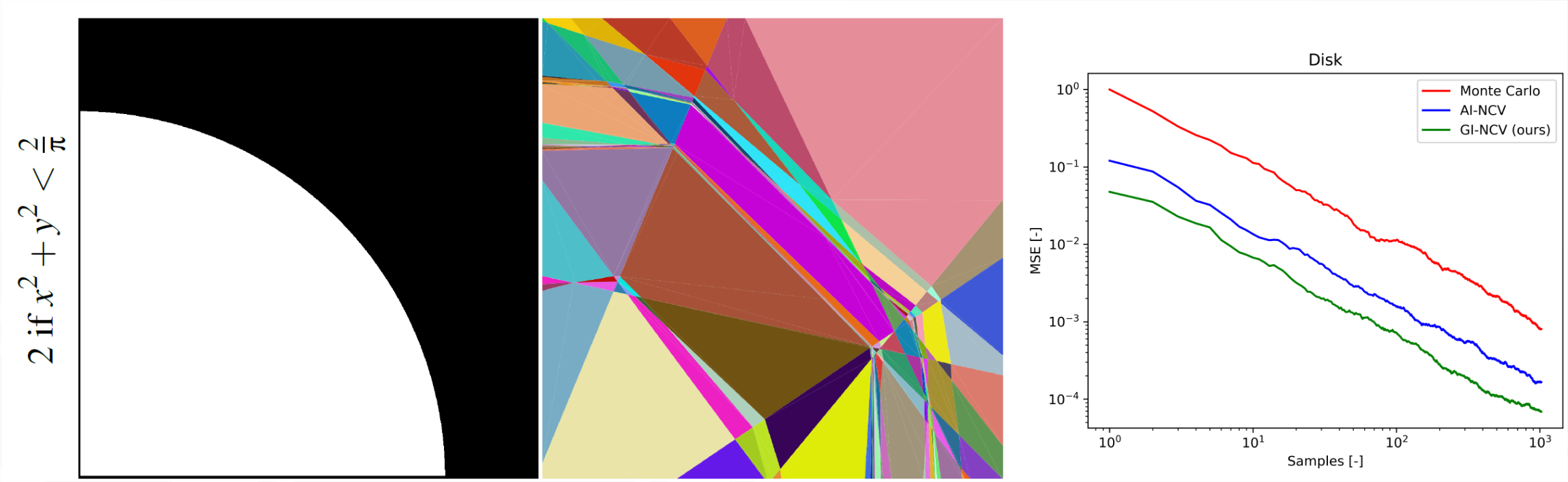


Applications: Analytic Functions

- Proof-of-concept implementation in PyTorch
- MLP with 2 hidden layers with 32 neurons each with ReLU
- Automatic integration [Li et al. 2024] as a reference method
  - Relying on the autograd in PyTorch
  - Sigmoid instead of ReLU

Disk Function

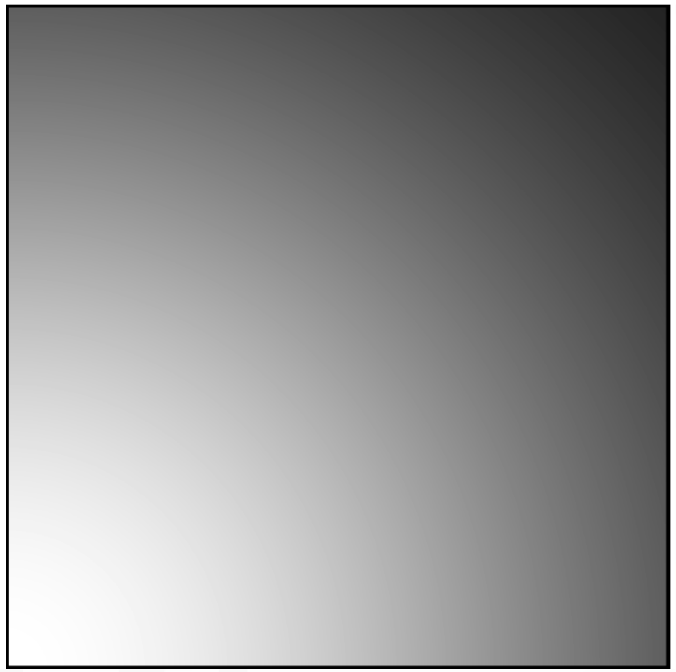
- Both methods are competitive
- The main difference is due to a different activation function

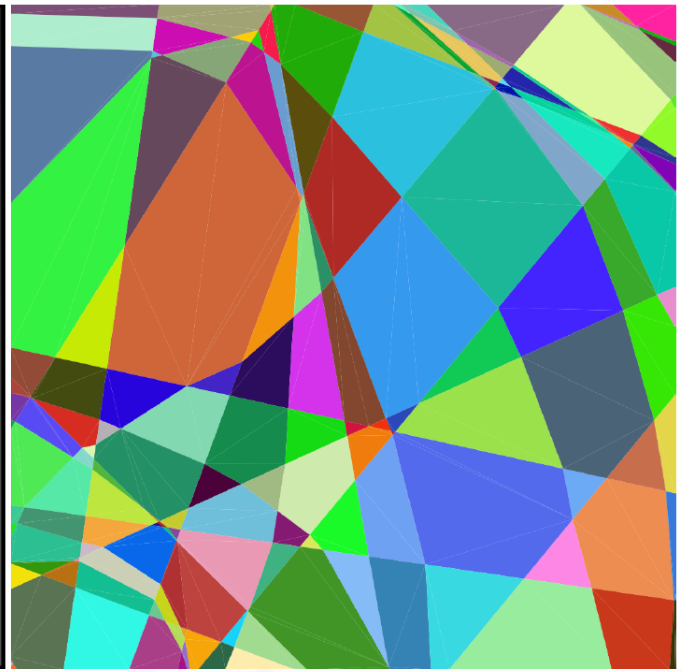


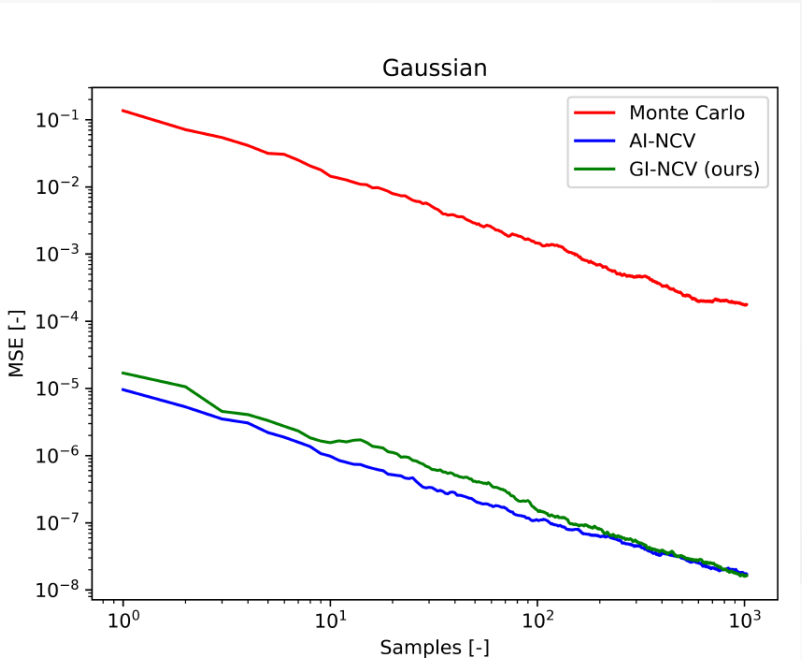
Gaussian Function

- Both methods are competitive
- The main difference is due to a different activation function

$$\frac{4}{\pi \mathrm{erf}^2(1)} \exp\left(-x^2 - y^2\right)$$







Applications: Light Transport

- Solving many integrals (e.g., one per pixel)
- A single neural network $g(x, y \mid \phi)$
  - 2 hidden layers with 32 neurons each
- Arbitrary additional inputs ϕ
  - Fixed w.r.t integration
  - Arbitrary encoding
- Integrating a 2D slice of a higher dimensional function

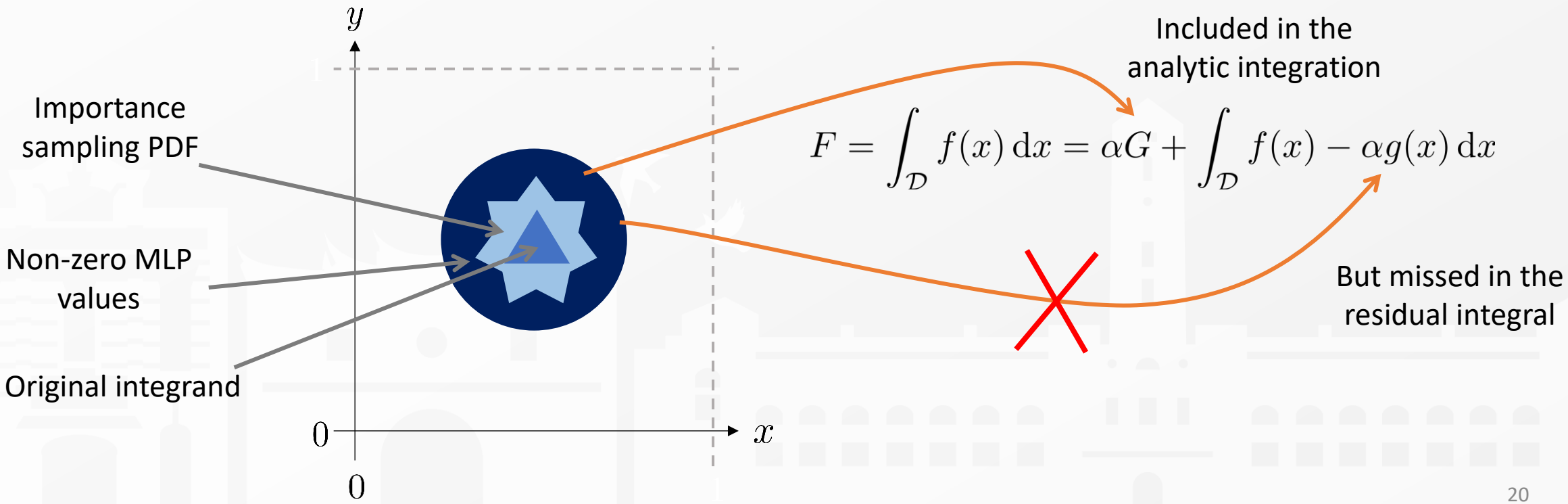
Parameter	Symbol	Encoding	Input dim.	Output dim.
Area light sample	(x, y)	Identity	2	2
Incoming direction	ω_i	Local hemisph. coords.	2	2
Position	$\mathbf{x}$	Hashgrid	3 (2)	2×4
Outgoing direction	ω_o	Sph. coords. / one-blob	2	2×4
Surface normal	$\mathbf{n}(\mathbf{x})$	Sph. coords. / one-blob	2	2×4
Diffuse albedo	$\sigma(\mathbf{x})$	Identity	3	3
Surface roughness	$\rho(\mathbf{x})$	Identity	1	1

Implementation Details

- Pre-train → Geometric Integration → Integrate residual integral using MC
- In-house HIP renderer & MLP framework
  - HIPRT 2.5 for ray tracing
  - Adam optimizer
  - Weight & biases in FP32
  - Exponential moving average for weights & biases
- Geometric integration as a separate kernel
  - Persistent threads to reduce memory requirements
- One training sample per pixel
- AMD Radeon PRO W7900 (48GB)

Control Variates and Importance Sampling

- Analytic integration domain must match integration of domain of the residual integral
- Probability density might be zero where the original integrand is zero
- MLP might provide arbitrary values, which are included in the analytic integration
- But these samples might be missed in the residual integral due to importance sampling



Ambient Occlusion

- Integration over upper hemisphere

$$AO(\mathbf{x} \mid r) = \int_{\omega_i \in \Omega} \frac{V(\mathbf{x}, \omega_i \mid r)(\mathbf{n}(\mathbf{x}) \cdot \omega_i)}{\pi} \, \mathrm{d}\omega_i$$

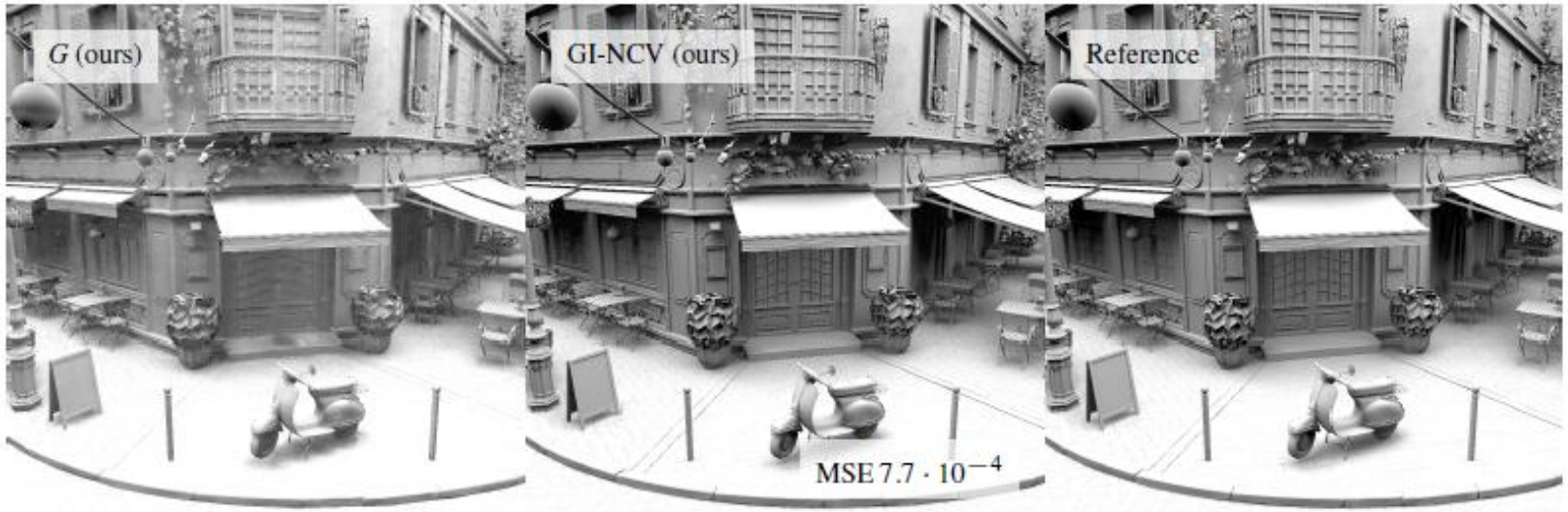
- Approximating integrand

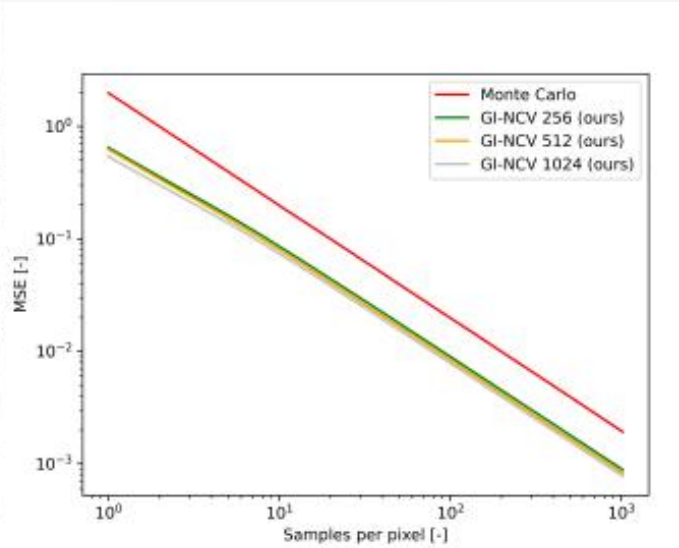
$$g(\omega_i \mid \mathbf{x}, \mathbf{n}(\mathbf{x})) \approx f(\mathbf{x}, \omega_i \mid r) = \frac{V(\mathbf{x}, \omega_i \mid r)(\mathbf{n}(\mathbf{x}) \cdot \omega_i)}{\pi}$$

- Direction in local hemispherical coordinates
  - Transform to the local space
  - Hemi-spherical coordinates (height & azimuth)

Ambient Occlusion - Bistro

Comparison to vanilla Monte Carlo using different number of training iterations





Direct Lighting

- Integration over an area light source

$$L_o^{direct}(\mathbf{x}, \omega_o) = \int_{\mathbf{y} \in A} f_r(\mathbf{x}, \omega_{\mathbf{x}\to\mathbf{y}}, \omega_o) L_e(\mathbf{y}, \omega_{\mathbf{y}\to\mathbf{x}}) G(\mathbf{x}, \mathbf{y}) V(\mathbf{x}, \mathbf{y}) \, \mathrm{d}\mathbf{y}$$

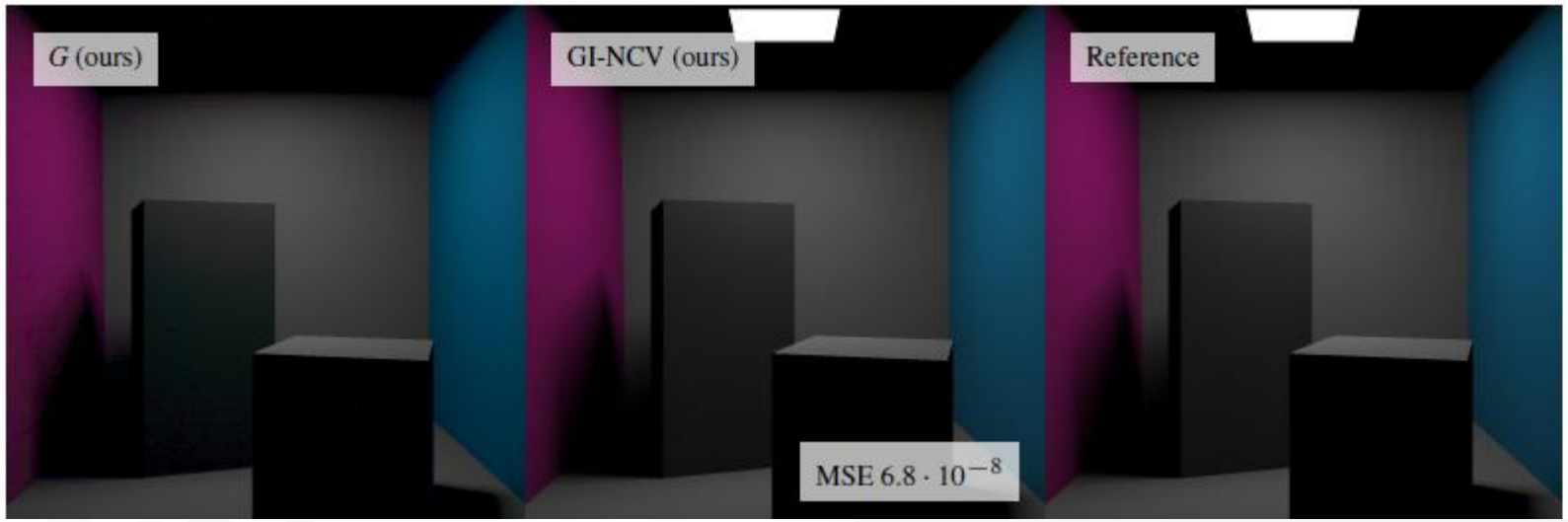
- Approximating integrand

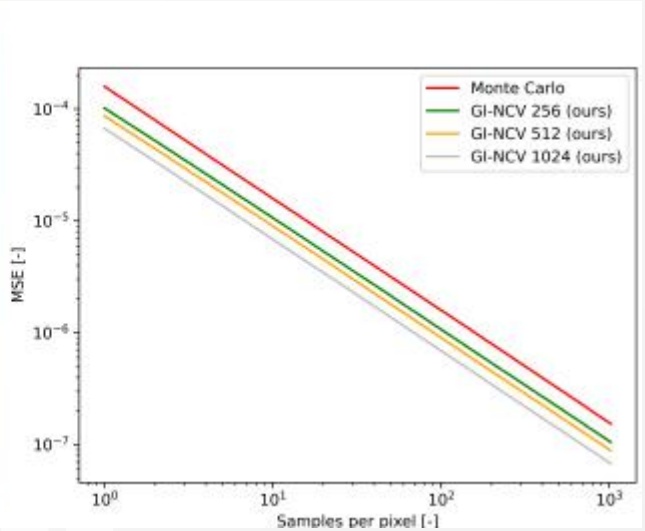
$$\mathbf{g}(x, y \mid \mathbf{x}, \omega_o, \mathbf{n}(\mathbf{x}), \sigma(\mathbf{x}), \rho(\mathbf{x})) \approx \mathbf{f}(\mathbf{x}, \mathbf{v} + x\mathbf{e}_1 + y\mathbf{e}_2, \omega_o) = \mathbf{f}(\mathbf{x}, \mathbf{y}, \omega_o)$$

$$\mathbf{f}(\mathbf{x}, \mathbf{y}, \omega_o) = f_r(\mathbf{x}, \omega_{\mathbf{x}\to\mathbf{y}}, \omega_o) L_e(\mathbf{y}, \omega_{\mathbf{y}\to\mathbf{x}}) G(\mathbf{x}, \mathbf{y}) V(\mathbf{x}, \mathbf{y}),$$

Direct Lighting – Cornell Box

Comparison to vanilla Monte Carlo using different number of training iterations





Global Illumination

- Integration over upper hemisphere

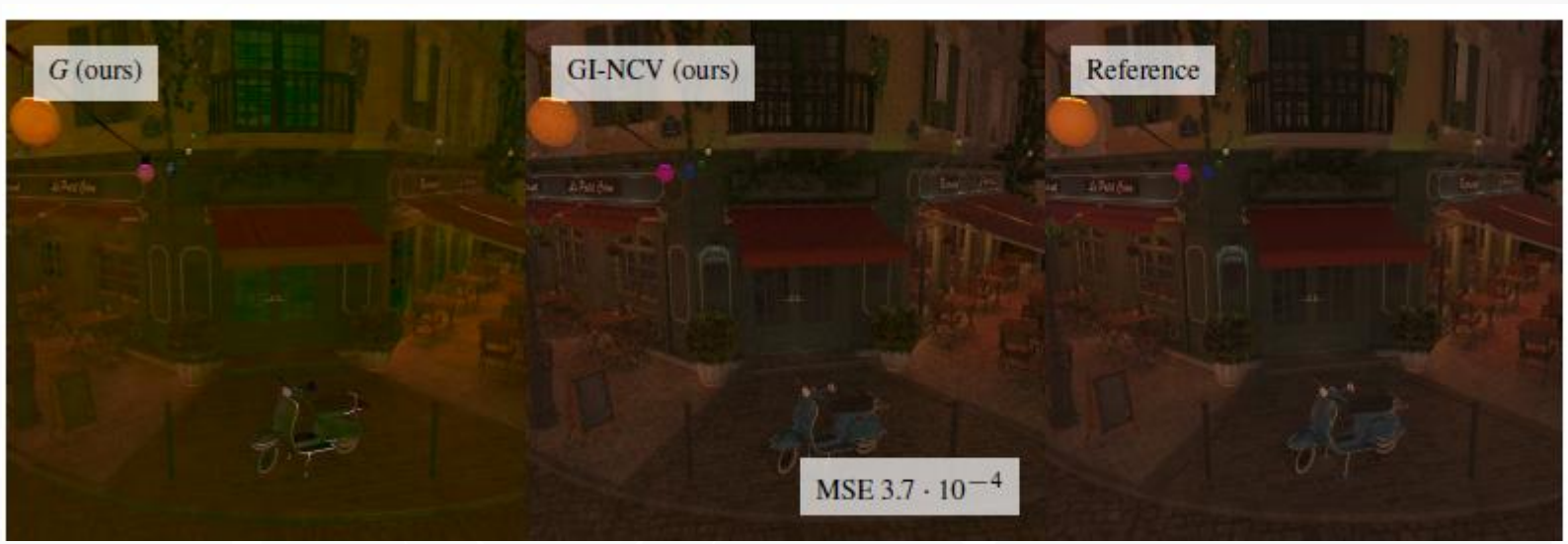
$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\omega_i \in \Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\mathbf{n}(\mathbf{x}) \cdot \omega_i) \, \mathrm{d}\omega_i$$

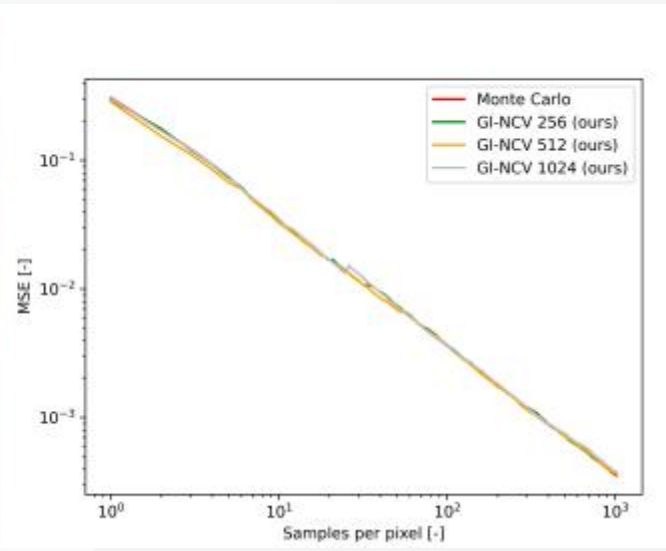
- Approximating integrand

$$\mathbf{g}(\omega_i \mid \mathbf{x}, \omega_o, \mathbf{n}(\mathbf{x}), \sigma(\mathbf{x}), \rho(\mathbf{x})) \approx \mathbf{f}(\mathbf{x}, \omega_i, \omega_o) = f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\mathbf{n}(\mathbf{x}) \cdot \omega_i)$$

Indirect Lighting – Bistro

Comparison to vanilla Monte Carlo using different number of training itertations



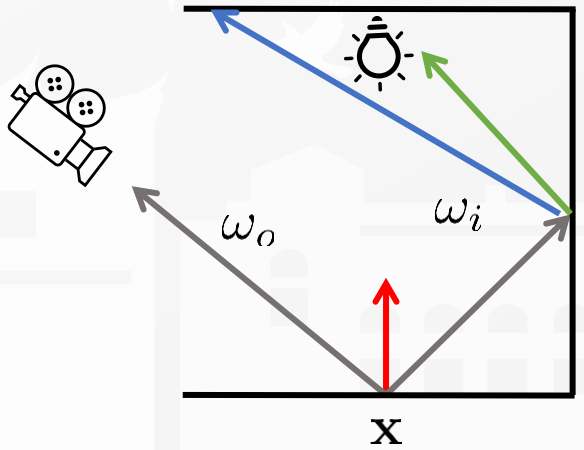


Even though the MLP approximate radiance field well,
control variates does not bring any improvement due to noisy estimates

Global Illumination – Noisy Estimates

- Approximating 7D function (position and two directions)
  - We need to train the whole integrand compared to *radiance caching*
- Notice that the ground truth data are not available
  - Incidence radiance is a nested integral itself
  - MLP can still learn the function from one-sample noisy estimates
  - Thanks to Adam optimizer and exponential moving average (tracking history ~ averaging)

$$\mathbf{g}(\omega_i \mid \mathbf{x}, \omega_o, \mathbf{n}(\mathbf{x}), \sigma(\mathbf{x}), \rho(\mathbf{x})) \approx \mathbf{f}(\mathbf{x}, \omega_i, \omega_o) = f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\mathbf{n}(\mathbf{x}) \cdot \omega_i)$$



The same argument but different function values

Noisy Estimates - Experiment

Integrating a simple function

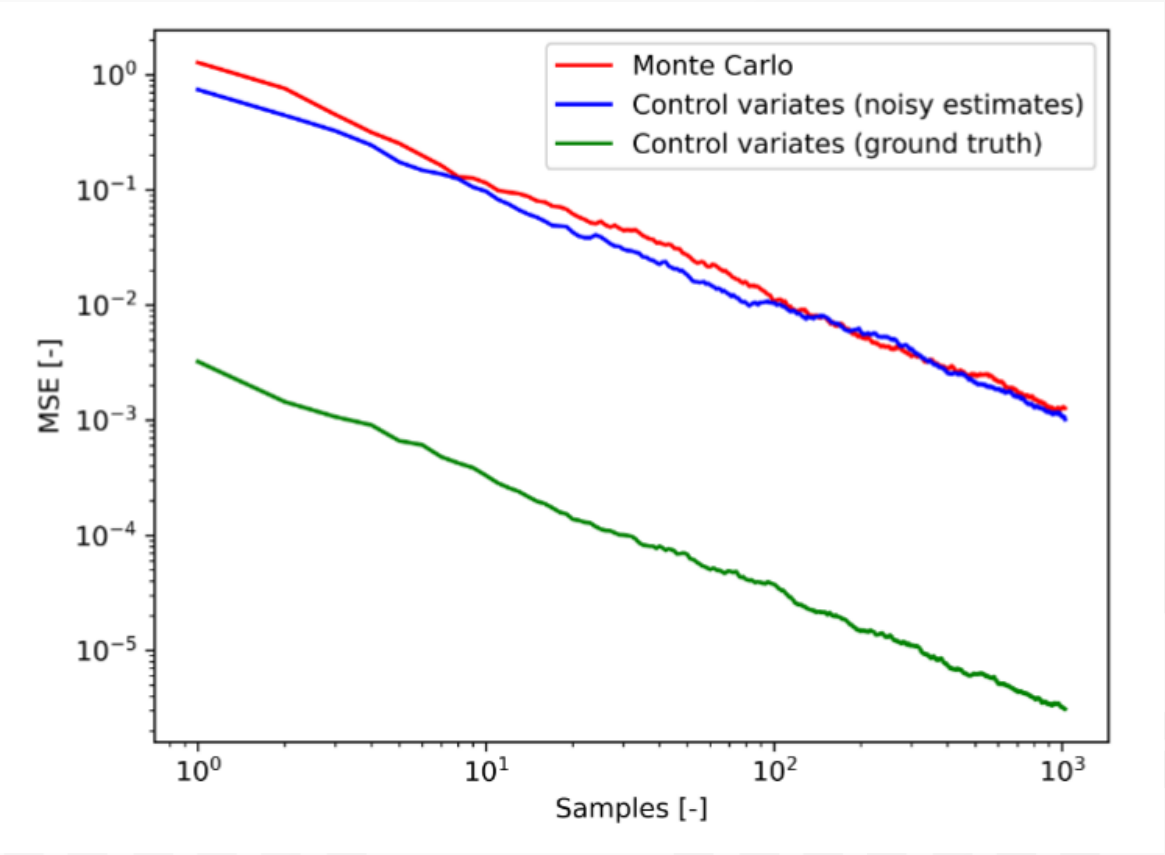
$$f(x) = \int_0^1 \int_0^1 8xyz \, \mathrm{d}y \, \mathrm{d}z = 2x$$

Control variate (almost perfect)

$$g(x) = 0.9 \cdot f(x)$$

Noisy estimates vs. ground truth values

$$\bar{f}(x) = 8xy_iz_i \approx f(x) = 2x$$



Discussion and Limitations

- Same as automatic integration limited to low dimensions (2D)
  - Cannot use full path tracing (with Russian roulette)
  - Possible to extend to higher dimensions in theory but it would be prohibitively expensive
- Same as other control variates methods difficult to use importance sampling
  - Analytic integration covers the whole integration domain (i.e., the unit square)
- Analytic integration is expensive (both in time and space)
  - Amortize the cost through many samples
  - Cannot use in real-time rendering
- Cannot use non-linear encoding for the integration domain
  - To keep the lines straight (not to become curvy)

	Monte Carlo	GI-NCV (ours)
SPP [-]	32768	1024 + 18432
MSE [-]	$4.54 \cdot 10^{-5}$	$4.53 \cdot 10^{-5}$
Render time [s]	136	78
Train time [s]	-	33
Integration time [s]	-	10
Inference time [s]	-	34
Total time [s]	136	155

Conclusion

- Analytic integration of MLP with piece-wise linear act. functions
  - Viable alternative to the automatic integration
- Integration formulated as geometric problem in 2D
  - Practical algorithm relying on tools from computational geometry
  - Looking at MLPs from a different perspective
- Applications in light transport
  - Reducing variance using control variates
  - Many limitations for practical use

Future work

- Importance sampling using the subdivision

Thank you for your attention!

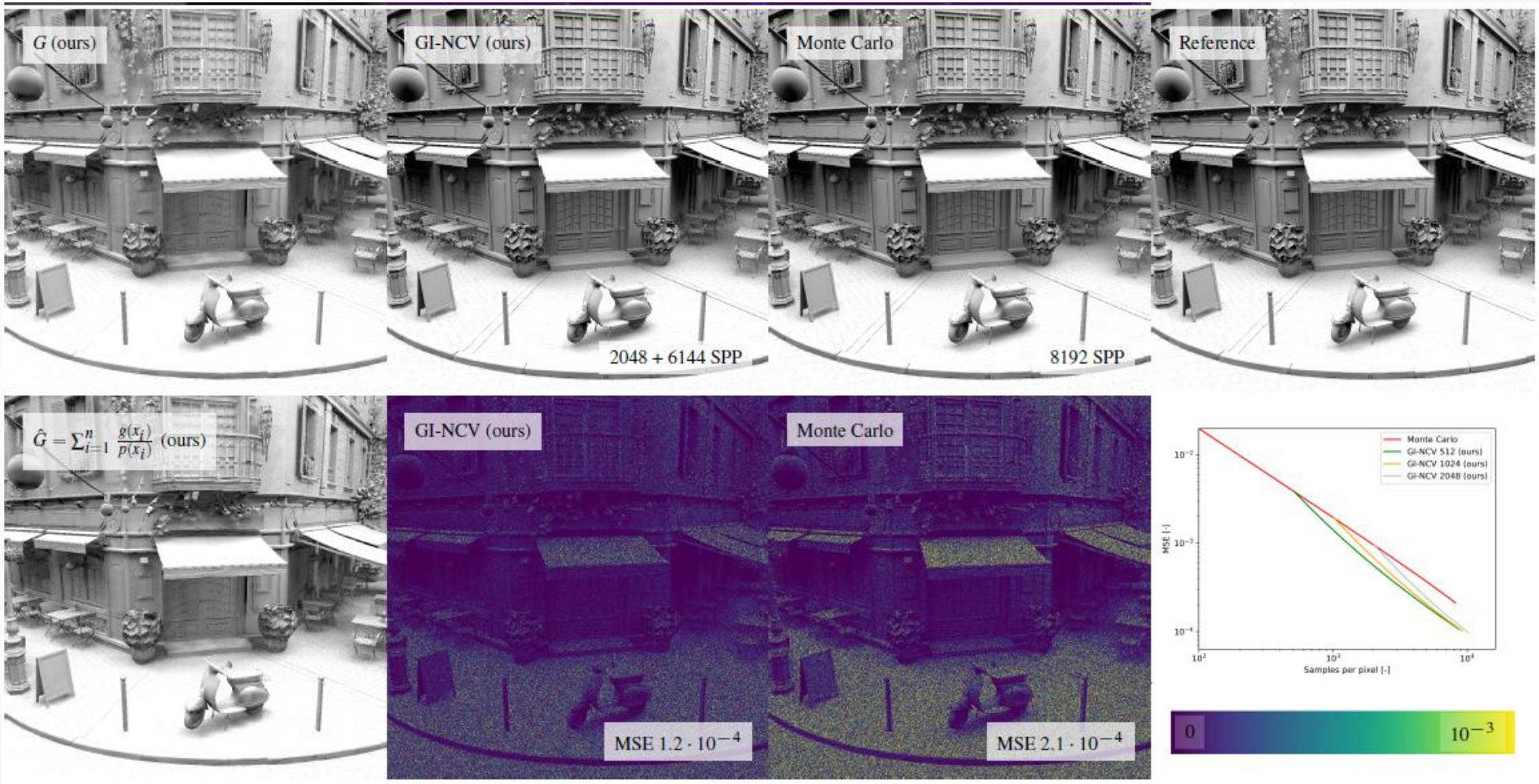
Backup Slides

The Number of Faces

- The particular number depends on the weights & biases
- Some lines might be completely outside the unit square
  - An does not introduce new faces

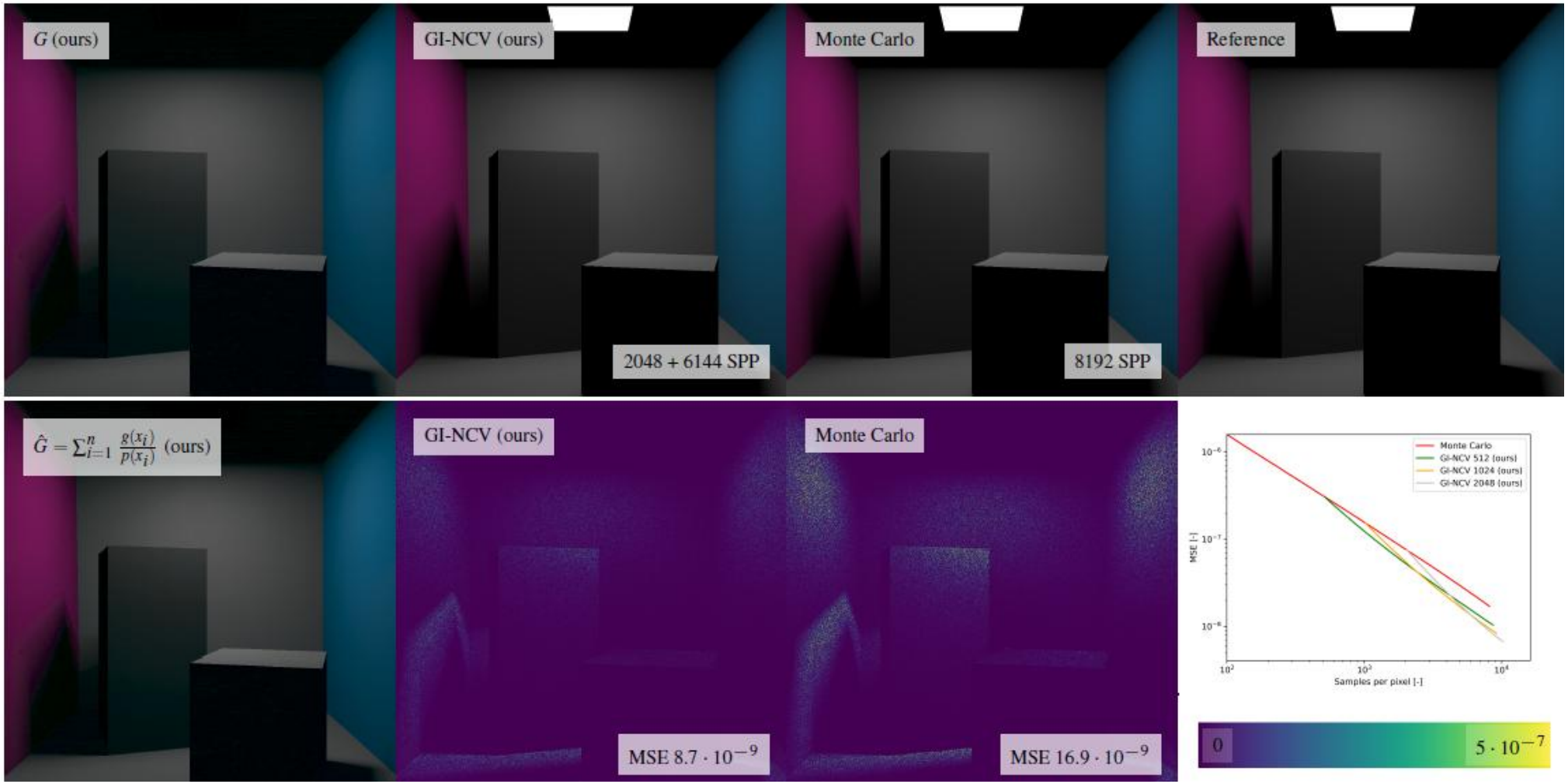
Ambient Occlusion - Bistro

Using the same samples for training and integration (correlated samples)



Direct Lighting – Cornell Box

Using the same samples for training and integration (correlated samples)



Correlated Samples

$$\langle I \rangle = \frac{m}{m+n}\langle I_{MC} \rangle + \frac{n}{m+n}\langle I_{CV} \rangle,$$

$$\langle I_{MC} \rangle = \frac{1}{m}\sum_{i=1}^{m}\frac{f(x_i)}{p(x_i)}$$

$$\langle I_{CV} \rangle = G + \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_{m+i}) - g(x_{m+i})}{p(x_{m+i})}$$

$$\begin{aligned}
\mathbb{E}[\langle I \rangle] &= \mathbb{E}\left[\frac{m}{m+n}\langle I_{MC} \rangle\right] + \mathbb{E}\left[\frac{n}{m+n}\langle I_{CV} \rangle\right], \\
&= \frac{mF}{m+n} + \frac{n}{m+n}\mathbb{E}\left[G + \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_{m+i}) - g(x_{m+i})}{p(x_{m+i})}\right], \\
&= \frac{mF}{m+n} + \frac{n}{m+n}\left(G + \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\frac{f(x_{m+i}) - g(x_{m+i})}{p(x_{m+i})}\right]\right), \\
&= \frac{mF}{m+n} + \frac{n}{m+n}(G + F - G) = \frac{(m+n)F}{m+n} = F.
\end{aligned}$$