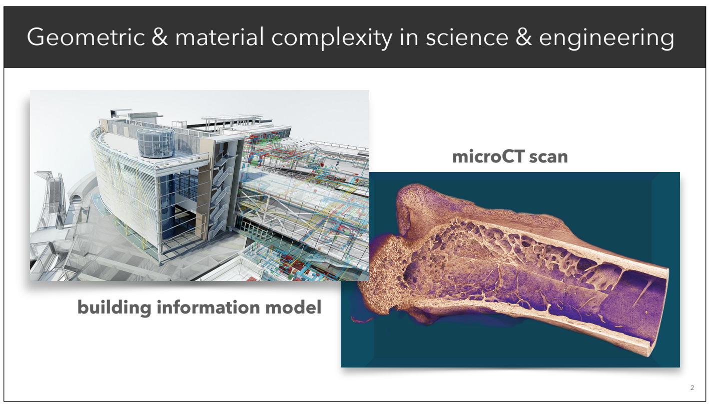
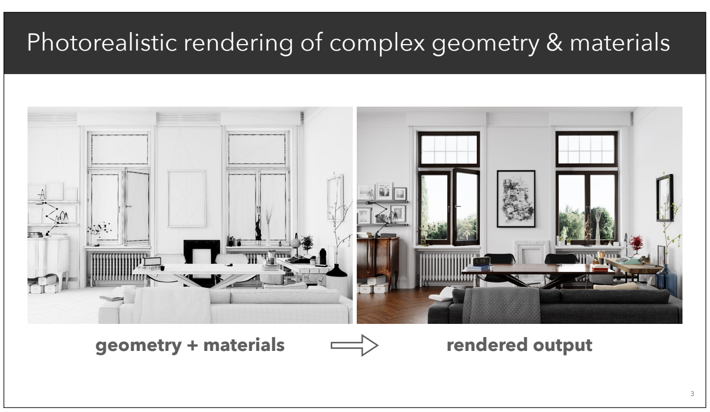
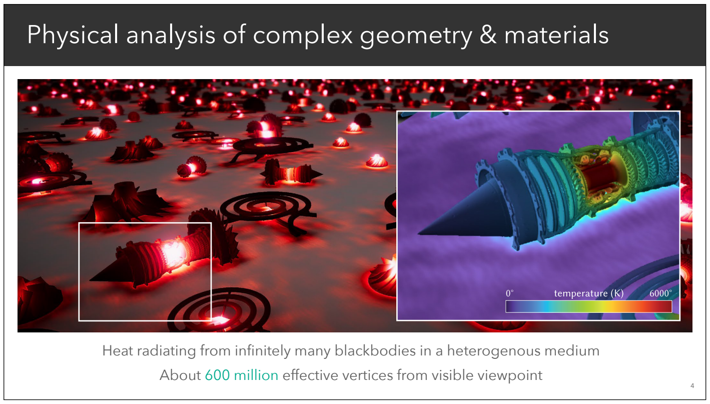
- Hi everyone, in this talk we'll present a Monte Carlo method to solve partial differential equations with spatially varying coefficients.



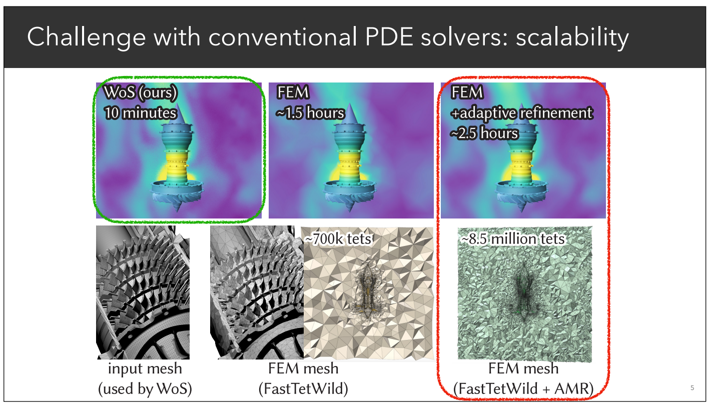
- Models in engineerings & science often have way more complexity in their geometry and materials than what conventional PDE solvers can handle.



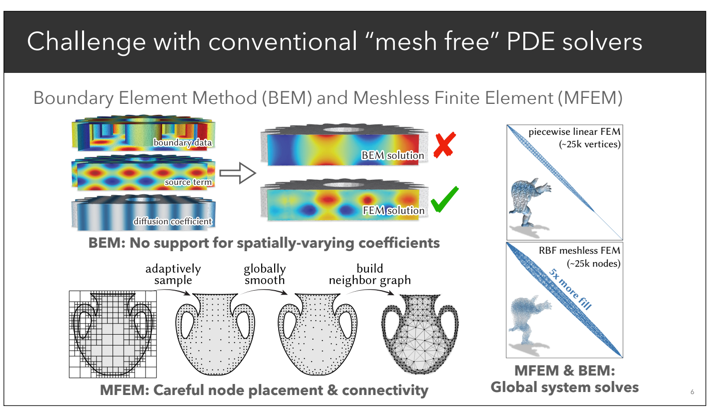
- But imagine if simulation was like Monte Carlo rendering: just load up a complex model and hit go without worrying about meshing or basis functions.
- Our paper takes a major step towards this vision by building a bridge between PDEs and volume rendering.



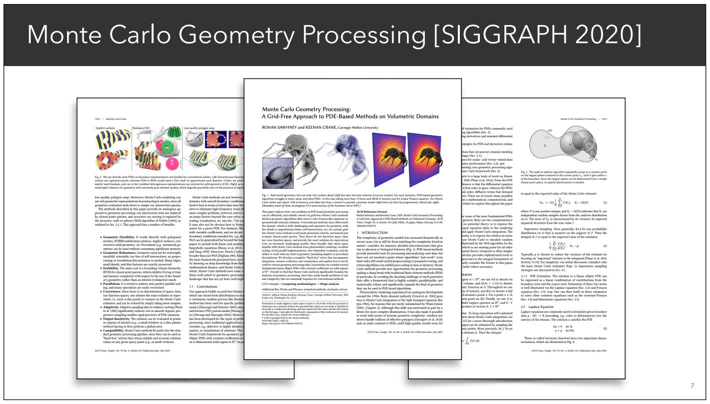
- Here’s an example: heat radiating off of infinitely many black body emitters, each with super-detailed geometry and material coefficients.
- From this view alone, the boundary meshes have ~600M vertices.
- To get the same level of detail with a conventional PDE solver such as



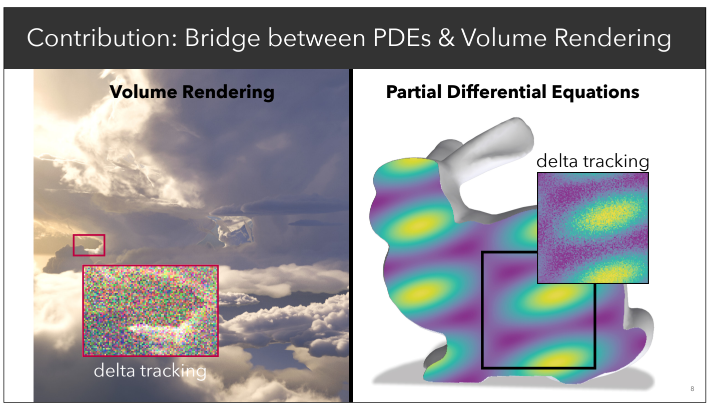
- the finite element method, we need about 8.5M tetrahedra & 2.5 hours of meshing time even on a tiny piece of the scene.
- But with Monte Carlo we get rapid feedback that can be progressively refined.



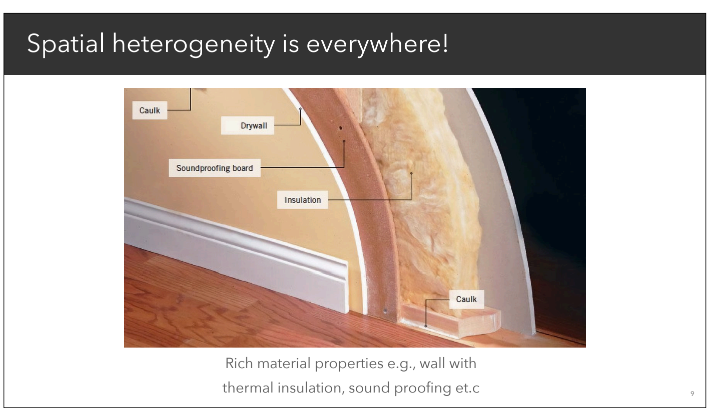
- At this moment, experts might point to the boundary element method and meshless FEM.
- The short story here is that these methods either lack support for variable coefficients, or they must still do expensive & error-prone node placement and global solves.



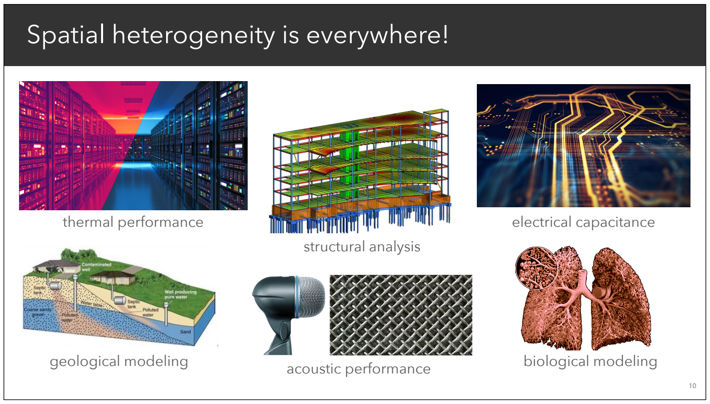
- Our journey with Monte Carlo began a few years ago, when we realized that rendering techniques from computer graphics could be used to turbo charge Muller's "walk on spheres" algorithm for solving PDEs.



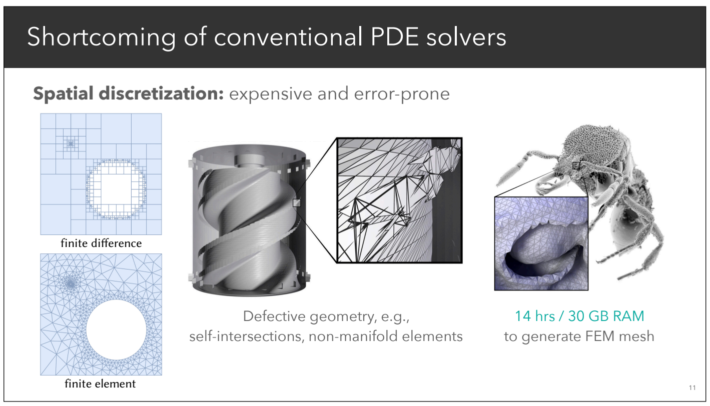
- In this paper, we make the connection between rendering & simulation even stronger, by linking and applying tools from volume rendering to PDEs with variable coefficients.



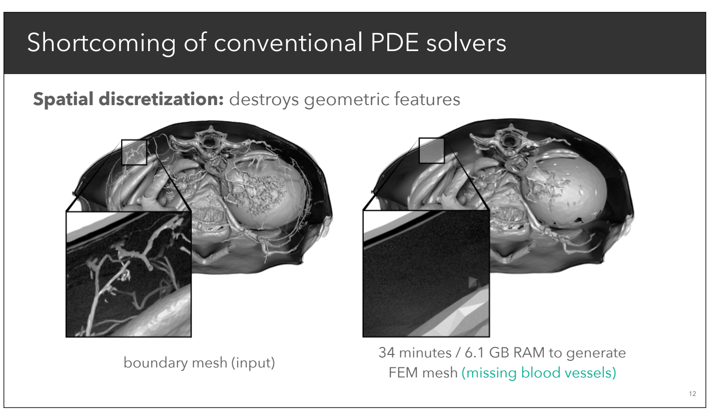
- Spatially-varying coefficients are essential for capturing rich material properties.
- For instance, to understand the thermal performance of a building, note that even a basic wall isn't just a homogeneous slab—it has many layers of different density.



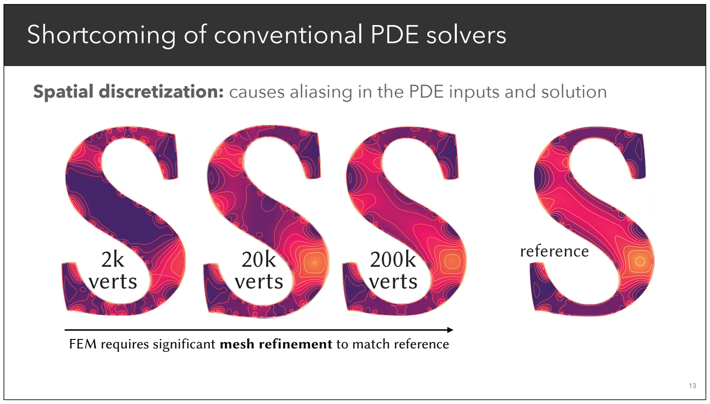
- And that's just the tip of the iceberg—PDEs with variable coefficients are everywhere in science & engineering, from thermal and structural analysis, to bimolecular and geological modeling.



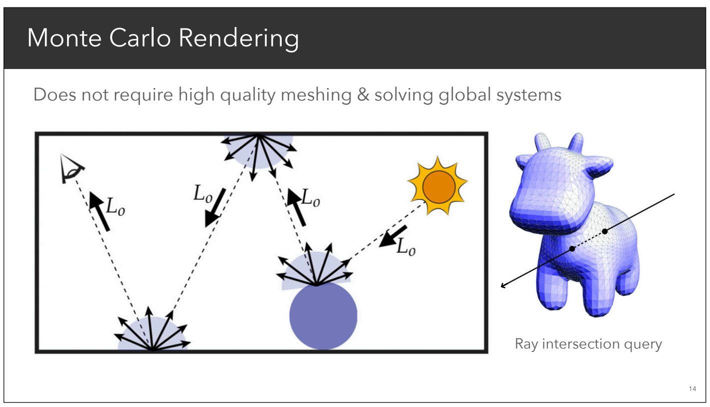
- A major challenge with any PDE solver is spatial discretization: this process is expensive and error prone, especially for complex geometric models.



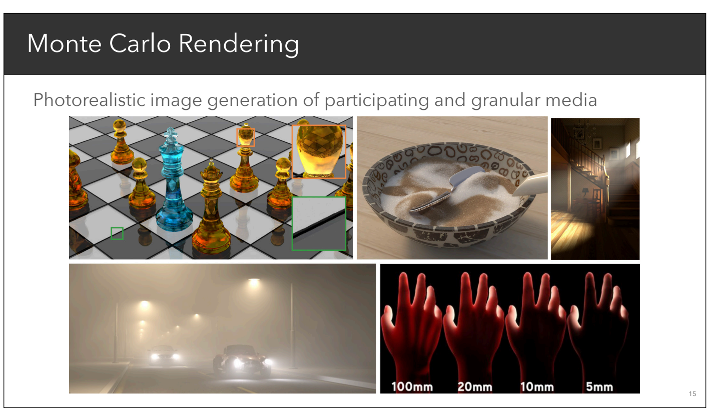
- Apart from its massive cost, discretization also causes two major headaches for solving PDEs:
- 1) important geometric features often get destroyed, and



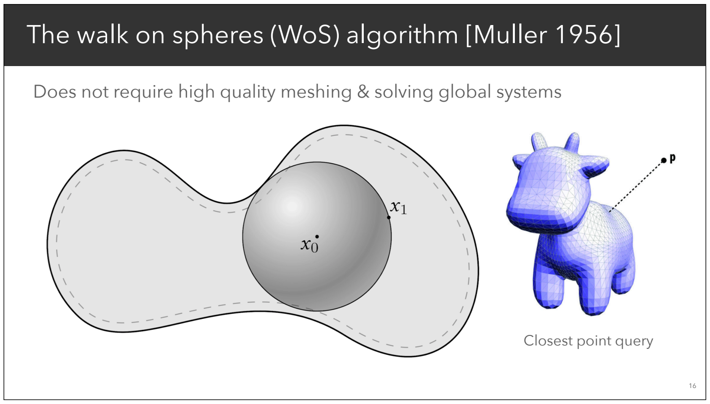
- 2) and significant mesh refinement can be needed to remove aliasing artifacts in the PDE solution, boundary conditions and coefficients.



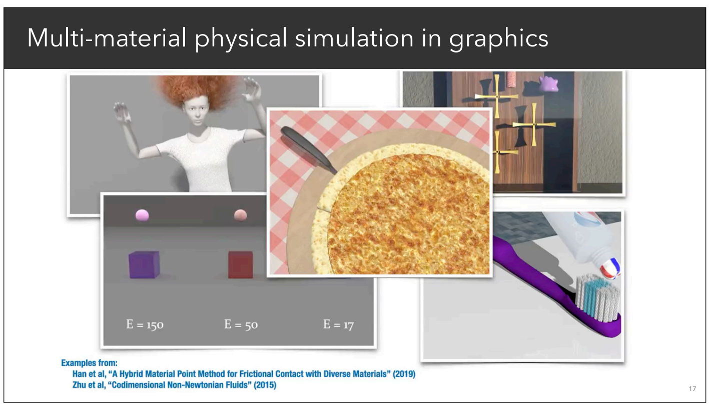
- To avoid these problems, photorealistic rendering moved away from meshing to Monte Carlo methods that only need point-wise access to the geometry via ray intersection queries.



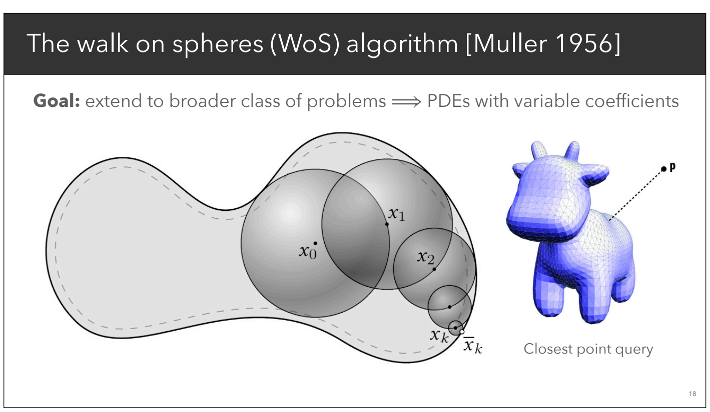
- This enabled simulation of intricate light transport phenomena on complex geometric models.
- So what about PDEs?



- Here there’s a little known algorithms called walk on spheres, which avoids spatial discretization altogether.
- Much like rendering, it only needs access to a single geometric kernel, namely closely point queries.
- Now, to be candid: WoS is *way* behind mature technology



- like FEM—especially in computer graphics we’ve seen amazing PDE solvers that handle complex multi-physics scenarios.

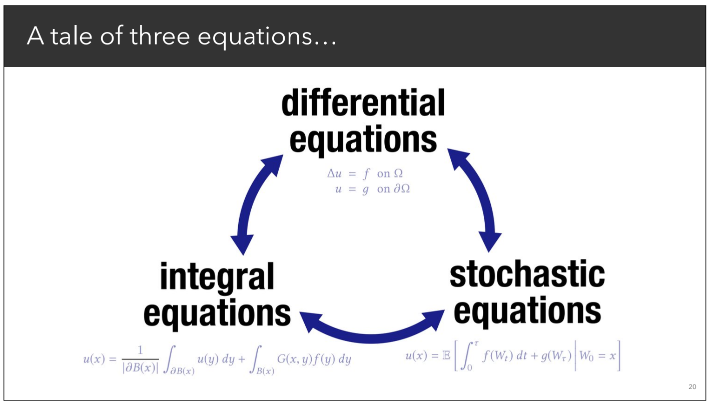


- But the WoS idea—and its promise to free us from the bonds for spatial discretization—is so appealing that we want to extend it to a broader classes of PDEs.
- So, let's talk about how

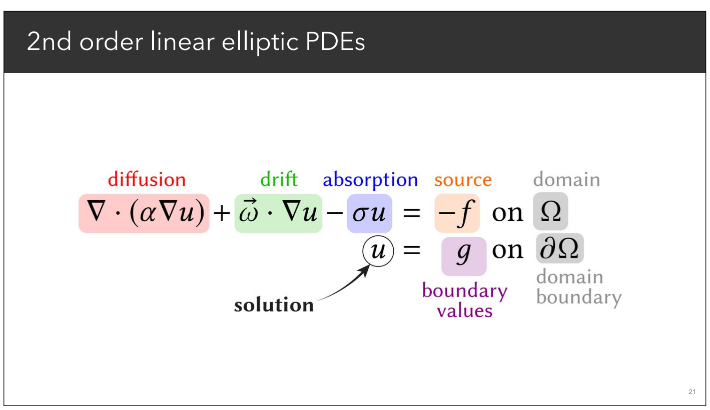
BACKGROUND

19

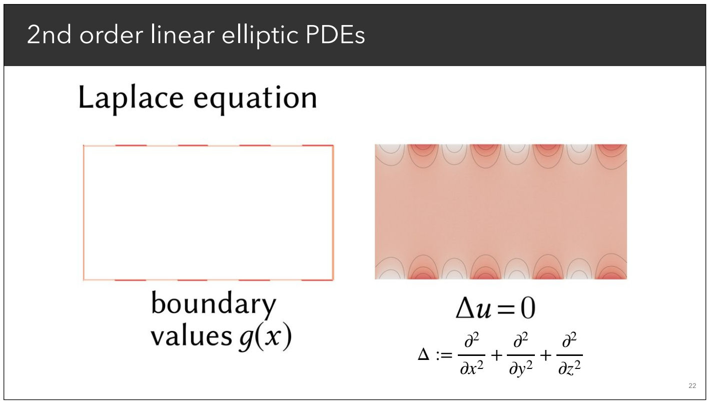
- we can extend WoS to variable-coefficient problems.



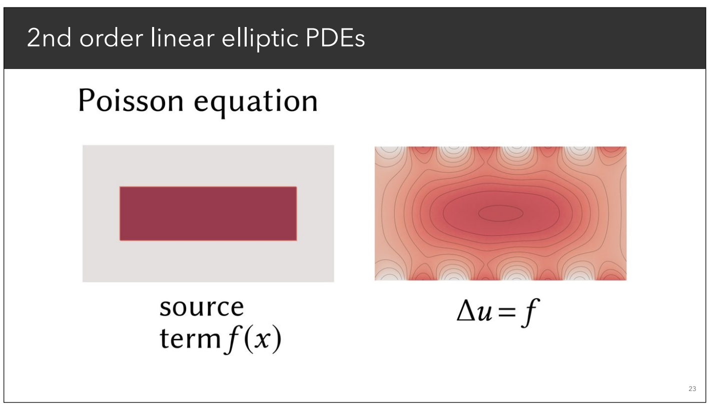
- This story is really a tale of three kinds of equations:
- 1) PDEs
- 2) integral equations and
- 3) stochastic differential equations.
- Our paper provides a playbook to convert between these different forms.



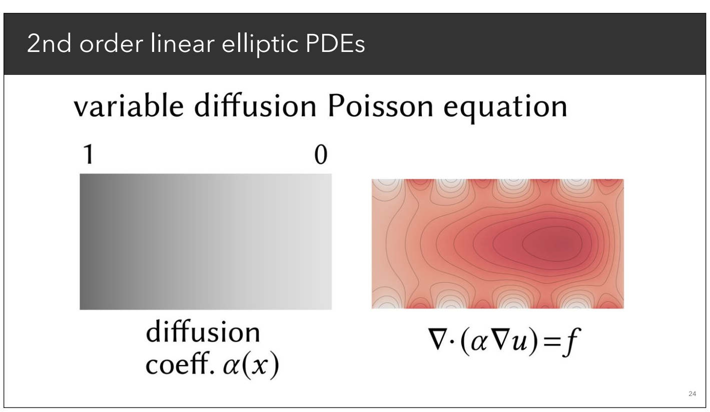
- In precise terms, our goal is to develop a Monte Carlo method that solves 2nd order linear elliptic equations with spatially-varying diffusion, drift, and absorption coefficients.
- Let's unpack this equation term by term.



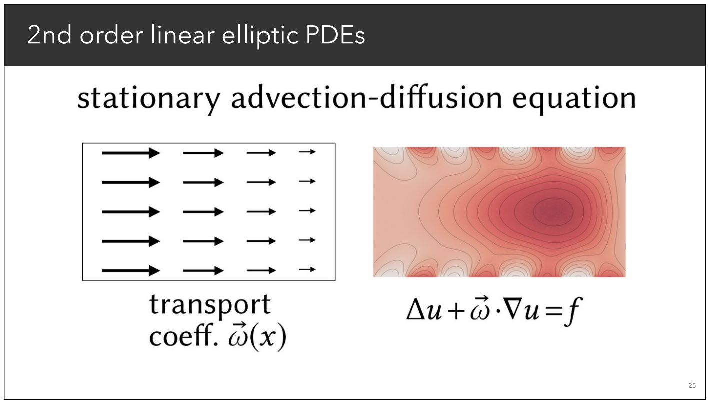
- First, a Laplace equation describes the steady-state temperature inside a domain if heat is fixed to some given function g on the boundary.



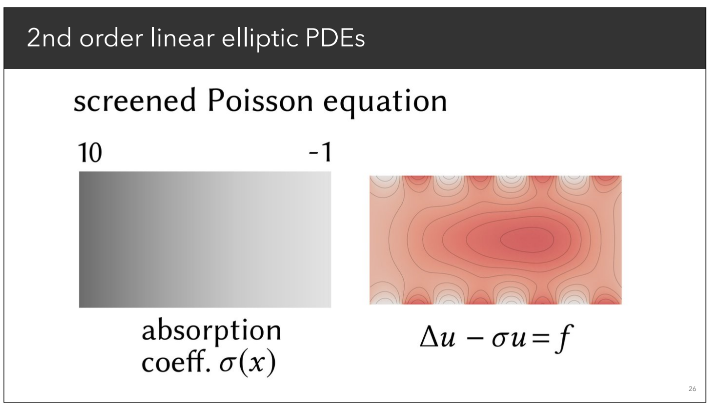
- Adding a source term f yields a Poisson equation, where f describes a background temperature.
- Imagine heat being pumped into the domain at a rate f at each point x.



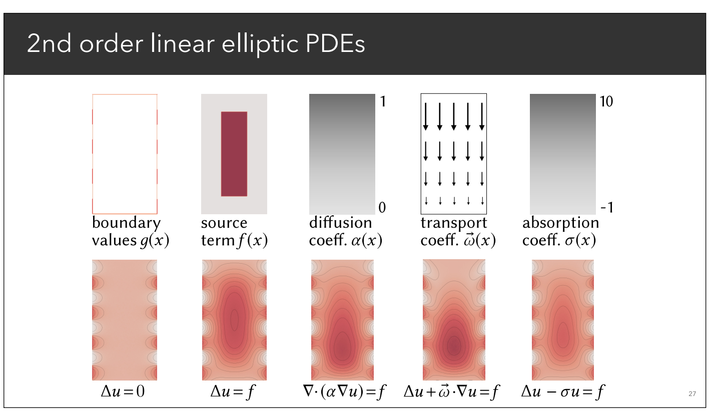
- We can control the rate of heat diffusion by replacing the laplacian with the operator \grad of (\alpha \grad u), where \alpha is a scalar function.
- Physically \alpha might describe the thickness or varying composition of a material.



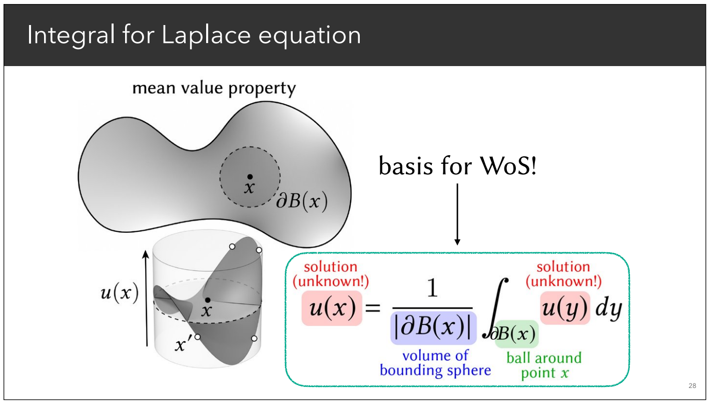
- Adding a drift term \omega \grad u to a Poisson equation indicates that heat is pushed along some vector field \omega—imagine a flowing river, which mixes hot water into cold water until it reaches a steady state.



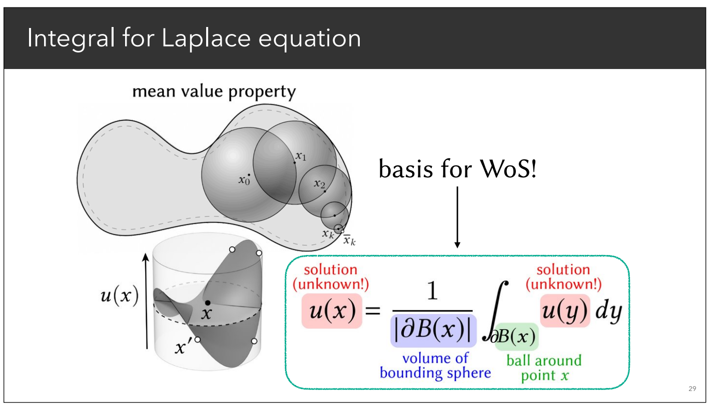
- Finally, an absorption term \sigma u acts like a background medium that absorbs heat—think about a heat sink or a cold engine block.
- The function \sigma describes the strength of absorption at each point x.



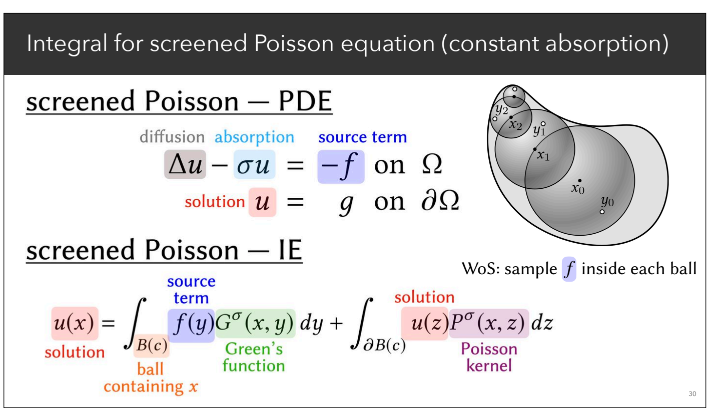
- There are lots of other terms you could add to a PDE, but these already get you pretty far.
- More importantly, these are terms we'll be able to convert into integral representations, and ultimately into Monte Carlo algorithms for PDEs!
- Let's see how…



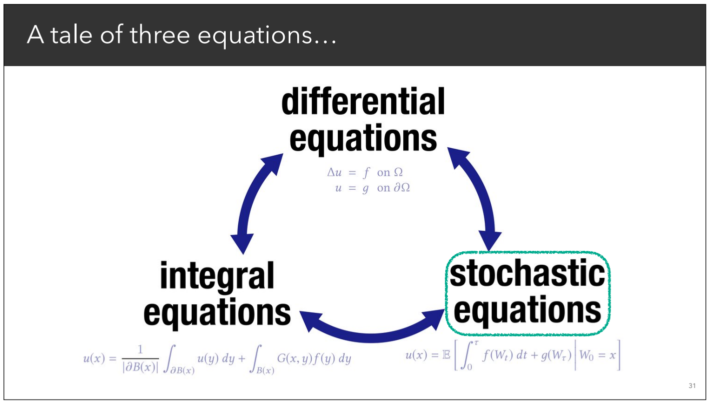
- The solutions to basic PDEs can be expressed via integral equations.
- E.g., the solution to a Laplace equation is given by the “mean value property“, which says that “the solution at a point x equals the average value over any empty ball centered at x”.
- This integral is *recursive*: the unknown value u at x depend on unknown values at y!
- Sounds like a problem, but this *exactly* how WoS works:



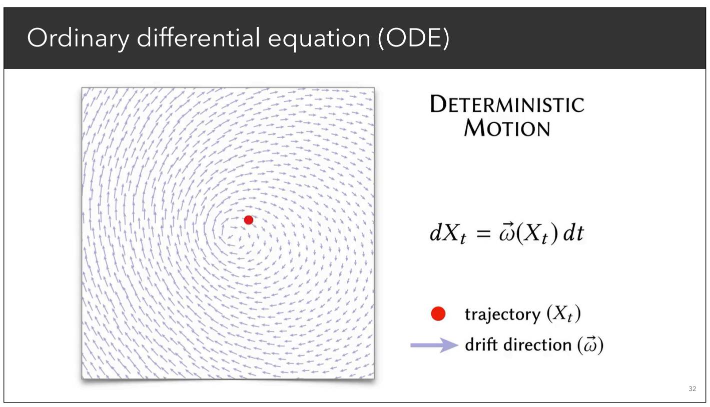
- recursively estimate the value of u till we reach the boundary and then grab the known boundary value.



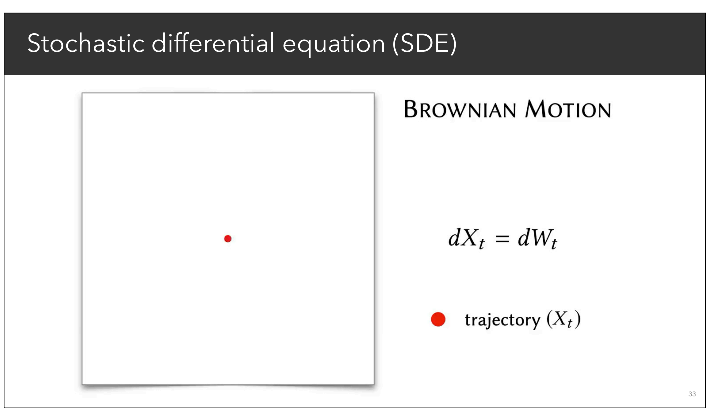
- As with PDEs, we can keep adding terms to integral equations to capture additional behavior.
- For instance there’s a nice integral representation for a screened Poisson equation, as long as the absorption coefficient sigma is constant!
- From an algorithmic perspective, WoS now also picks a random point inside each ball in the walk to sample the source term.



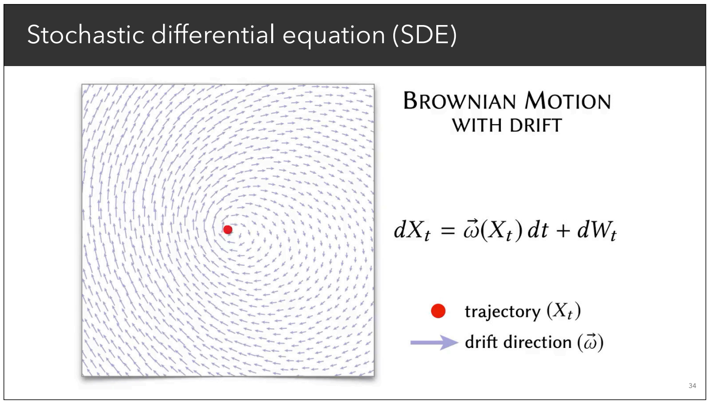
- Finally, we can describe the same phenomena using stochastic differential equations (SDE).
- For us, the stochastic picture is super important because it lets us deal with spatially-varying coefficients.



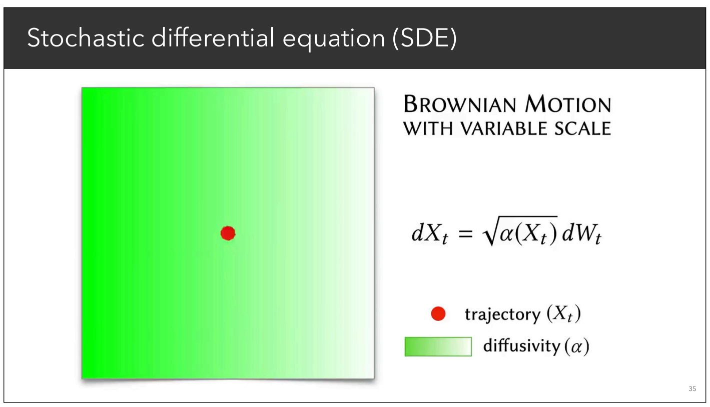
- An ODE describes the location of a particle in terms of its derivatives in time.
- For instance, dX = \omega dt says that a particle's velocity is given by some vector field \omega—e.g., a speck of dust blowing in the wind.



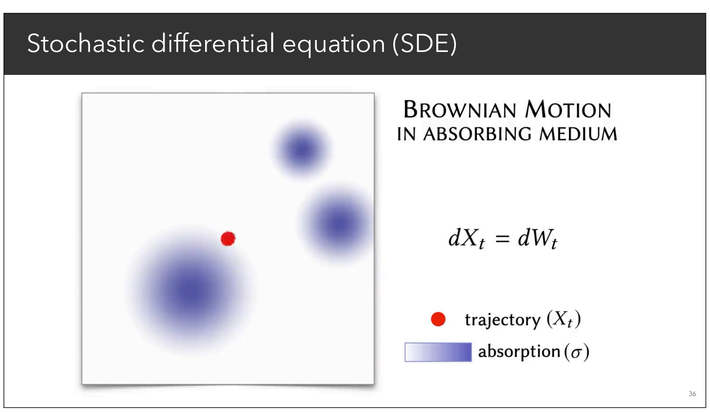
- A *stochastic* differential equation describes random motions—a key example is a Brownian motion, where changes in position follow a Gaussian distribution, and are independent of past events.
- Brownian motion is often used to model everything from moving molecules to fluctuations in stock prices.



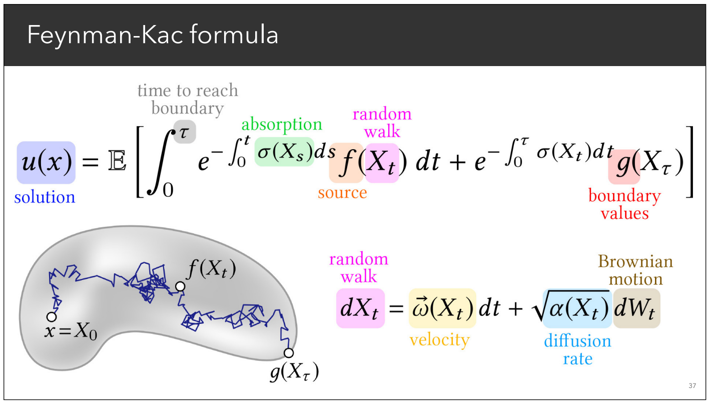
- Adding Brownian motion to our earlier ODE gives a more general *diffusion process* which we can think of as either a deterministic particle with noise, or a random walk with drift.



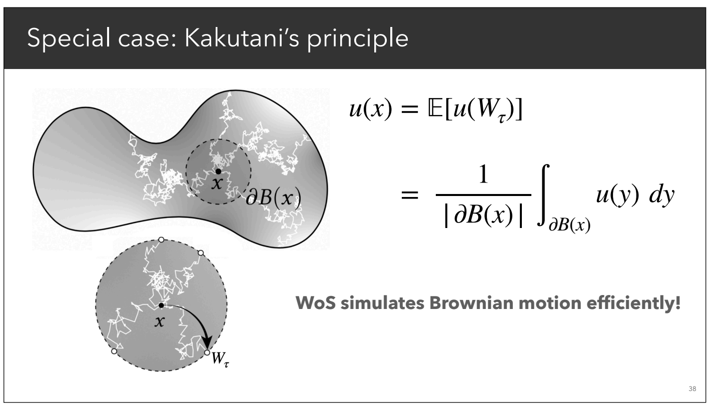
- We can also modulate the strength of the jiggling via a function \alpha. As \alpha increases, things “heat up”, and particles move faster.



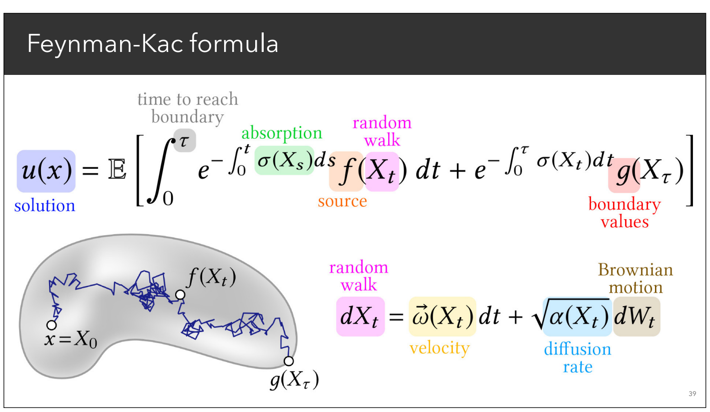
- Finally we can think about a random walker possibly getting absorbed in a background medium, like ink getting soaked up in a sponge.
- Here \sigma denotes the strength of absorption—it doesn’t show up in the SDE itself, but will be incorporated in a moment.



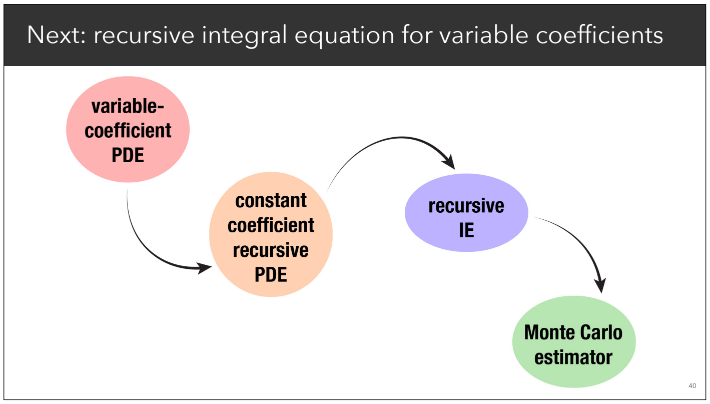
- Now, it's no coincidence that we use the same symbols \alpha, \sigma, \omega for both our PDE and SDE.
- These perspectives are linked by the Feynman-Kac formula, which gives the solution to our main PDE as an expectation over many random walks.



- A special case of Feynman Kac is Kakutani's principle, which says the solution to a Laplace equation is the average value seen by a Brownian random walk when it firsts hits the boundary.
- When restricted to a ball, Kakutani's principle is equivalent to the mean value property due to the rotational symmetry of Brownian motion.
- WoS can therefore be seen as an acceleration strategy for simulating brownian motion.

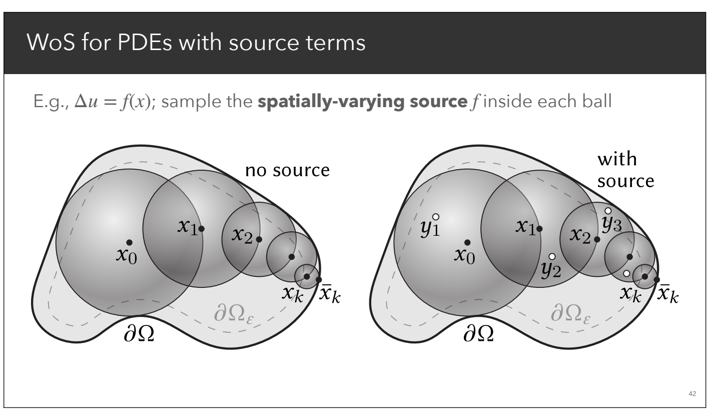


- More importantly, unlike classic integral equations, the Feynman-Kac formula handles spatially-varying coefficients!
- As a result, we can use it to

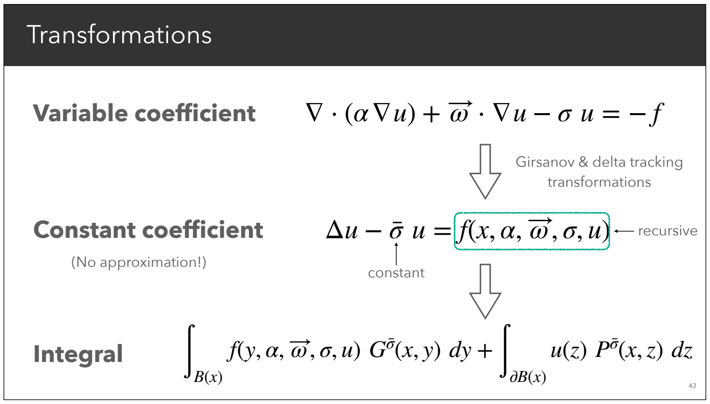


- build a *new* recursive but deterministic integral equation, which in turn leads to modified WoS algorithms for variable coefficient PDEs.

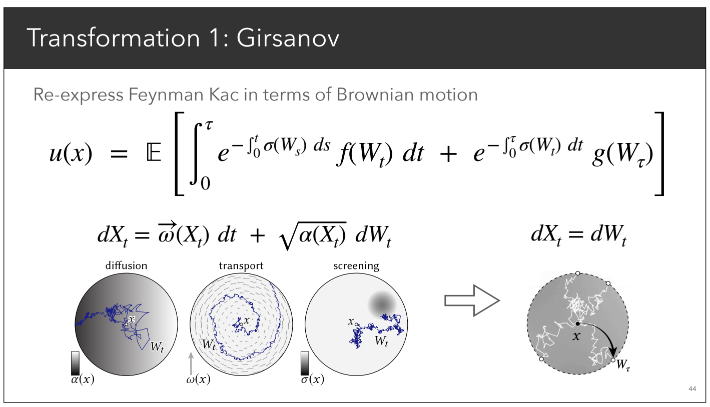
- The key observation behind our method



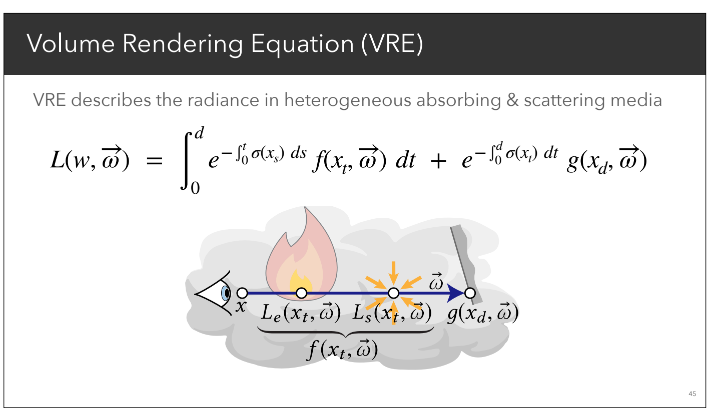
- Is that even though WoS cannot directly handle PDEs with variable coefficients, it can still be used to solve problems with spatially-varying source terms.



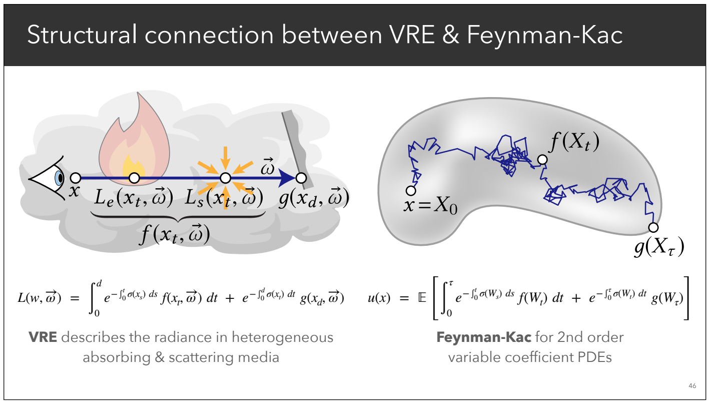
- We therefore apply a series of transformations that convert our original heterogeneous PDE into a *constant*-coefficient screened Poisson equation with a recursive source term.
- From an FEM perspective, it might feel like we haven't done anything useful: we just shuffled all the hard stuff to the other side of the equals sign.
- Yet from the Monte Carlo perspective we now have a way forward, since we can recursively estimate the resulting deterministic integral.



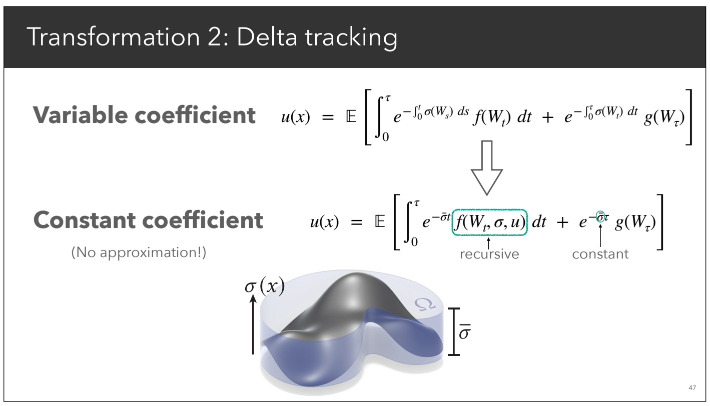
- On the stochastic front, our first transformation rewrites the Feynman Kac formula purely in terms of brownian motion instead of a diffusion process.
- As part of this transformation, all the original coefficients get converted into a single variable absorption coefficient \sigma.
- To get rid of this \sigma,



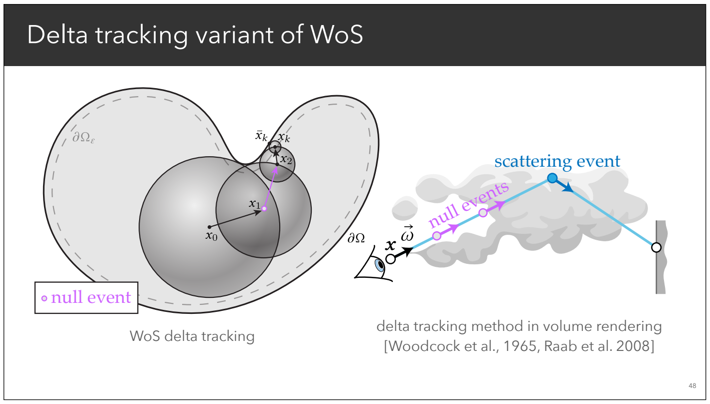
- we then observe that the Feynman Kac formula actually looks a lot like the volume rendering equation, which in computer graphics describes the radiance L along a ray in a heterogeneous medium that absorbs, scatters and emits radiation.
- But if for a second we put aside the physical meaning of these symbols,



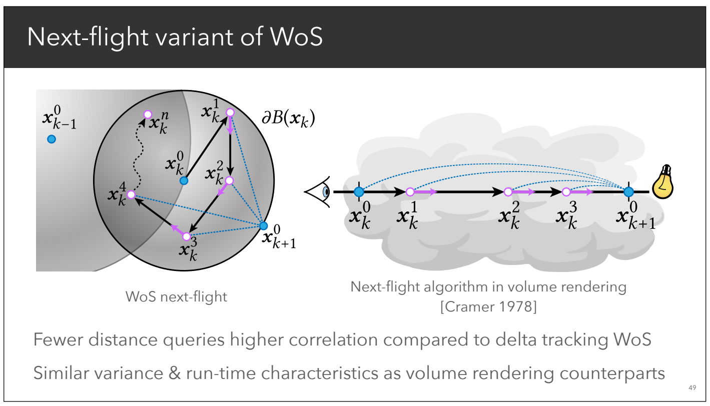
- then structurally the main difference between Feynman Kac and the VRE is that one requires simulation of Brownian random walks, while the other provides the radiance along a ray.
- And as a result, transformations like delta tracking used in graphics to solve the VRE



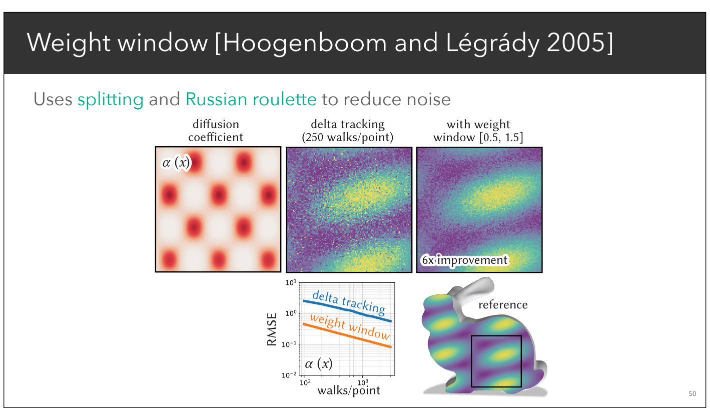
- can be applied to the Feynman Kac formula as well.
- Here we basically move the variable coefficient \sigma to a recursively defined source term f as in the PDE setting.
- \sigma \bar is a free parameter, which we set to the difference between the maximal values of \sigma over the entire domain.
- Just as in volume rendering, we've essentially turned our original heterogeneous medium into an equivalent homogeneous one.
- Now algorithmically, this is all really interesting because PDEs can suddenly benefit from decades worth of rendering research!



- In particular, these transformations allow us develop modified versions of WoS with direct counterparts in volume rendering.
- For instance, the delta tracking version, shown here on the left, uses the concept of null events from volume rendering to sample points either inside or on the boundary of a ball.



- Similarly, the next flight version always jumps to a random point on the largest sphere using off-centered walks, and conceptually it looks a lot like the next-flight algorithm from volume rendering.
- In practice, we find that while the next flight version requires fewer distance queries, it usually suffers from higher correlation compared to delta tracking.
- Both algorithms also share the variance and run-time characteristics of their volume rendering counterparts.



- Finally, on problems with high frequency coefficients, standard variance reduction techniques in Monte carlo rendering, like Russian roulette and splitting, can provide similarly dramatic improvements to our algorithms, here providing a 6x speedup.
- Run-time performance also improves in this case, since walks are often terminated early.

Implementation & Results

- From an implementation perspective,

PDE inputs

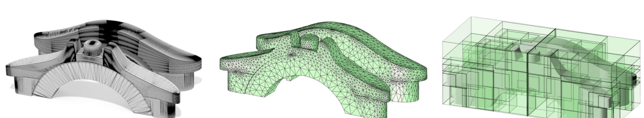
$$\overbrace{\nabla\cdot(\alpha\nabla u)}^{\text{diffusion}} + \overbrace{\vec{\omega}\cdot\nabla u}^{\text{drift}} - \overbrace{\sigma u}^{\text{absorption}} = \overbrace{-f}^{\text{source}} \text{ on } \overbrace{\Omega}^{\text{domain}}$$

$$\underbrace{u}_{\text{solution}} = \underbrace{g}_{\text{boundary values}} \text{ on } \underbrace{\partial\Omega}_{\text{domain boundary}}$$

- A PDE is encoded by the description of the scene geometry, boundary conditions, source term and coefficients.
- In our implementation, this data is provided via callback routines that return a value for any query point in the domain.

Acceleration of closest point queries

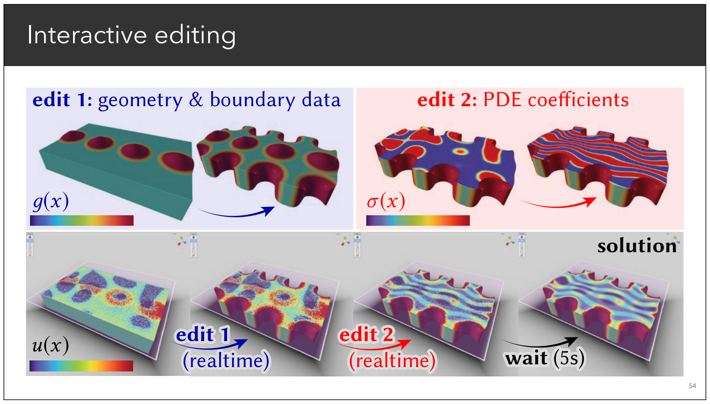
Accelerate closest point queries using BVH



input (Thingi10k #996816)	mesh w/ FastTetWild 1 hour 25 minutes	build BVH for WoS < 1 second

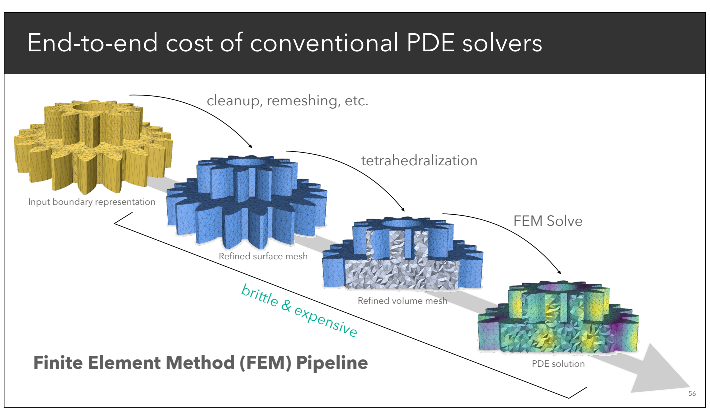
Unlike bad meshes, BVHs do not impact correctness/accuracy of PDE solution!

- Closest point queries can be accelerated via standard spatial hierarchies such as a BVH for a wide variety of scene representations.
- Unlike mesh generation, a BVH uses little memory and can be built quickly even for detailed models.
- Also, unlike a bad mesh, a poorly-constructed BVH only harms performance—not correctness or accuracy.

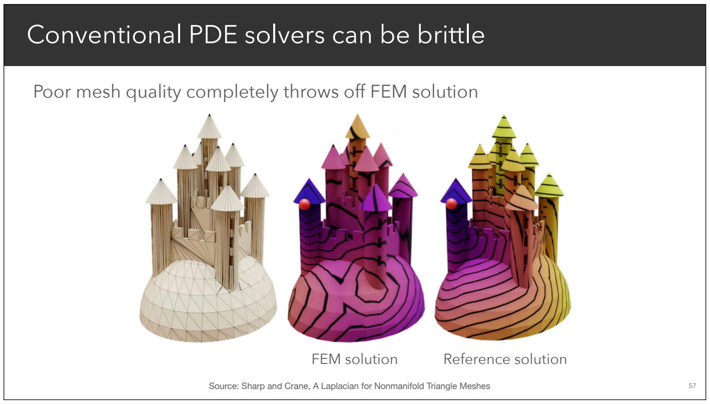


- Our approach is ideal for interactive editing since it operates directly on the original scene representation, and provides instant feedback after updates to the geometry, boundary conditions and PDE coefficients.

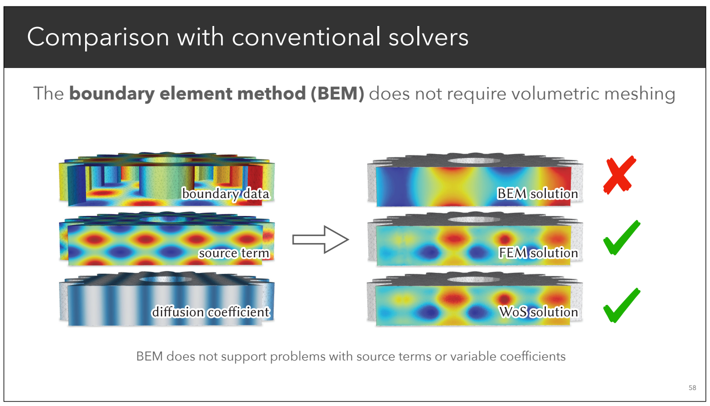
- Unlike conventional solvers, our method also doesn't require any geometric pre-processing, which allows it to scale to extremely large scenes.



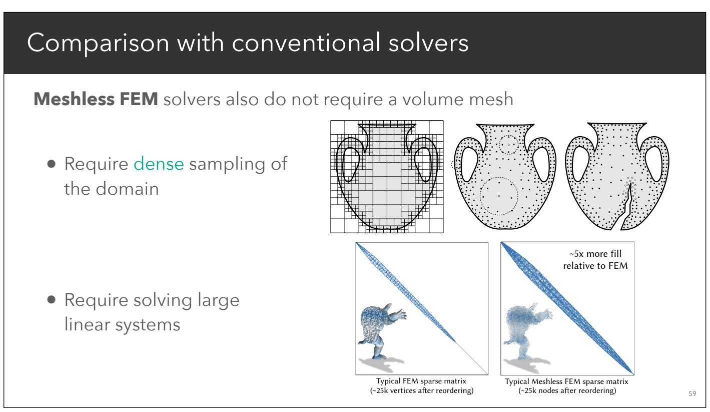
- In contrast, the significant issue with traditional numerical methods such as FEM is the end-to-end cost of the pipeline: even if the FEM solve is fast, one has to first convert the boundary description into a high quality simulation mesh.
- This process can be brittle and slow, and requires careful consideration since



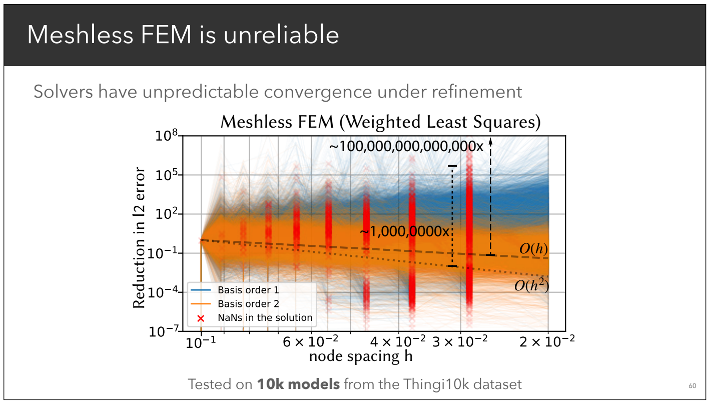
- even a single bad quality element can throw off the the accuracy of an FEM solution completely.



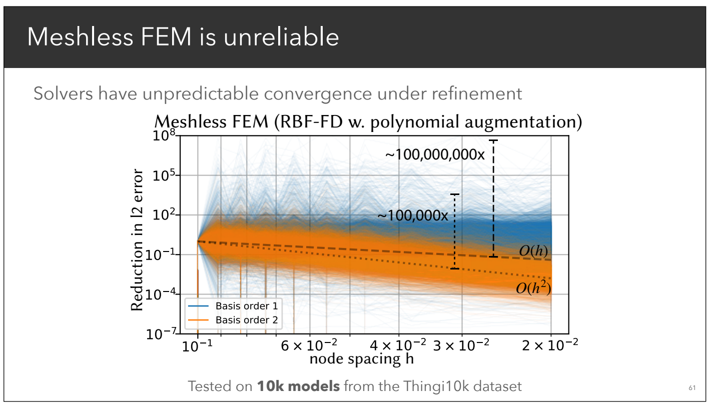
- Some conventional solvers such as the boundary element method don't need to mesh the domain.
- However, BEM can't handle problems with source terms or spatially-varying coefficients on the domain interior.
- To include these terms, it has to be coupled with a second solver such as FEM which requires volumetric meshing.



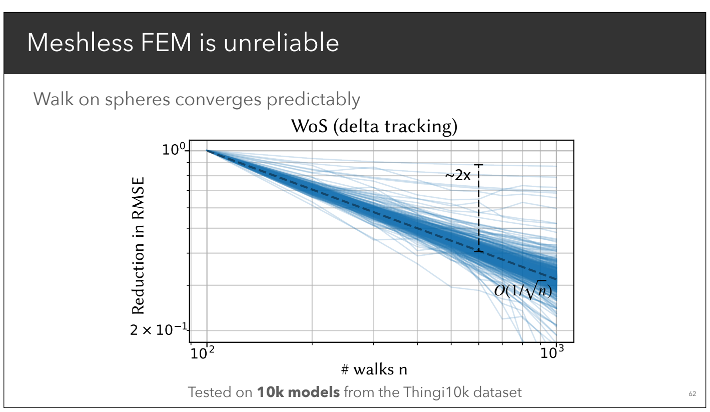
- Meshless FEM methods such as moving least squares also don't require meshing either.
- However, unlike Monte Carlo, these methods still require a dense and careful sampling of the entire domain.
- They also need to solve global systems of equations which are typically a lot larger in size compared to FEM.



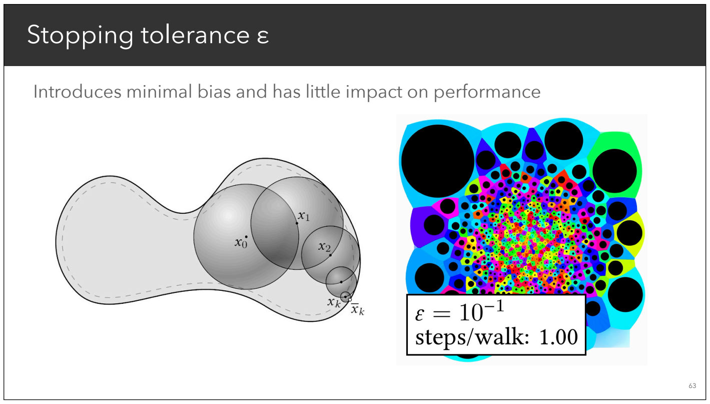
- The bigger problem with meshless methods is that they are unreliable.
- Here we solve a standard PDE on all models from the Thingi10k dataset, and plot the error under refinement.
- As indicated by the crosses in the plot, common meshless schemes often fail to converge under refinement, while



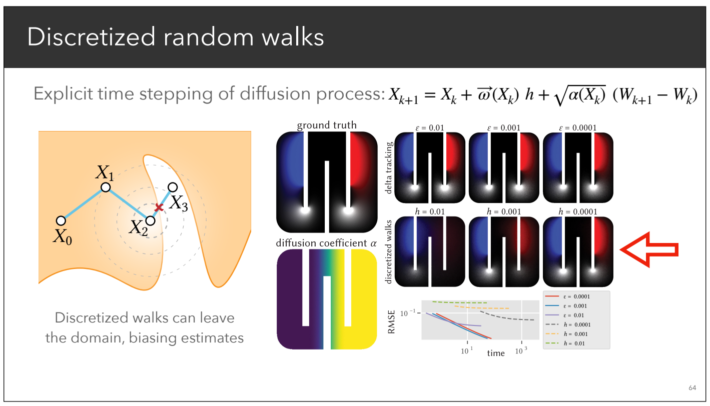
- even state of the art approaches show extremely large variation in error.



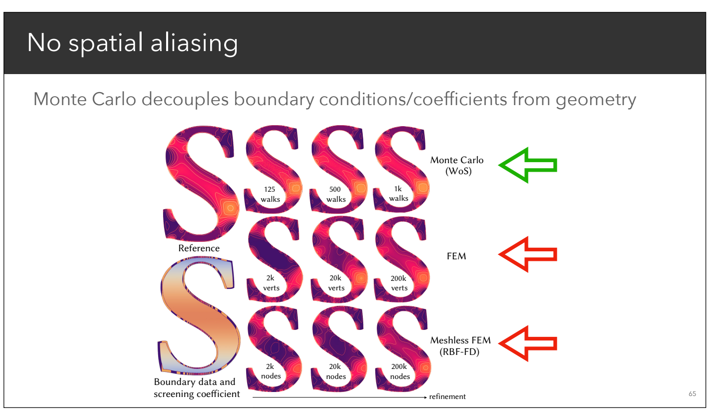
- In contrast, WoS demonstrates very predictable convergence on all 10000 models in the same dataset.



- Like standard WoS, the only parameter in our algorithms is an epsilon tolerance that specifies how close to the boundary we have to be before we can grab the known boundary value.
- This tolerance introduces minimal bias and has little impact on performance unlike tolerances in meshing algorithms.



- Like ray marching, Feynman Kac can be directly approximated by simulating a diffusion process with explicit time stepping.
- Unlike WoS however, discretized walks can leave the domain, which biases the solution estimates.
- Smaller time steps help reduce this bias, but at significant detriment to run-time performance.



- An extra benefit of Monte Carlo is that it decouples the boundary conditions and coefficients from the geometry.
- As a result, there is never any spatial aliasing, and WoS is able to capture the global profile of the solution with just a few walks.
- In contrast, conventional methods have to heavily refine the discretization to capture high frequency inputs.
- In general, it’s very difficult to predict an adequate mesh size ahead of time.

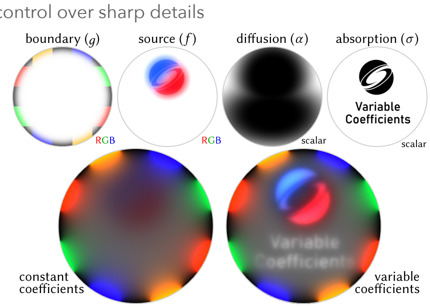
Physical analysis of complex geometry & materials

No homogenization of PDE coefficients!

- Though our main goal here is to develop core knowledge, there are some cool things we can immediately do.
- One is to simply solve physical PDEs with complex geometry and coefficients.

Example application: variable coefficient diffusion curves

Additional control over sharp details

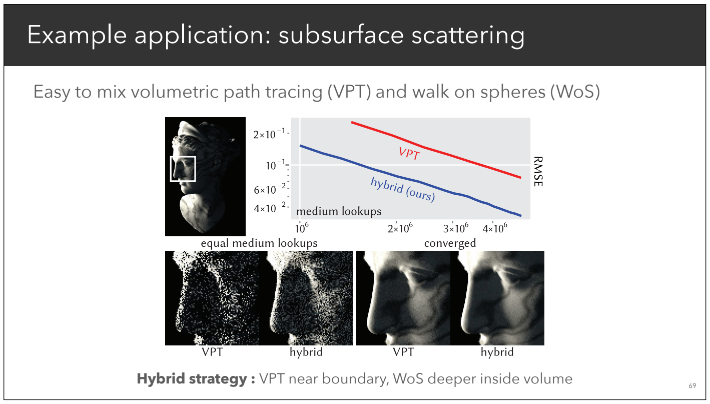


- A graphics example is to generalize so-called "diffusion curves" to variable coefficients, giving more control over how sharp or fuzzy details look.

Example application: diffusion curves on surfaces

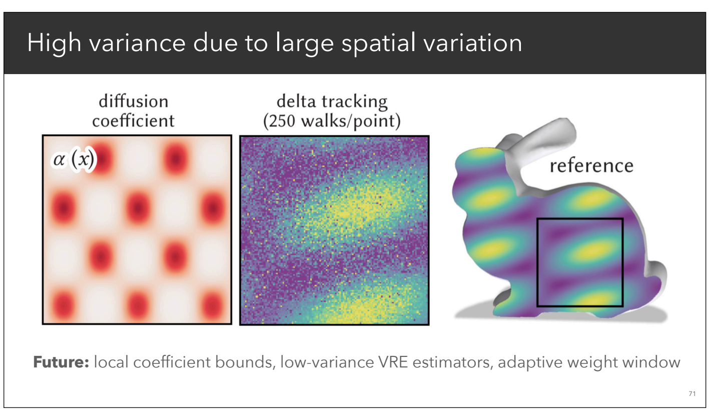
Use variable coeffs on flat domains to model constant coeffs on curved domains

- Another nice point of view is that variable coefficients on a flat domain can actually be used to model constant coefficients on a curved domain!
- This way, we can solve PDEs with intricate boundary data on smooth surfaces, without any meshing at all.



- A Monte Carlo method also makes it easy to integrate PDE solvers with physically-based renderers.
- For instance, we can get a way more accurate diffusion approximation of heterogeneous subsurface scattering, without having to painfully hook up to a FEM or grid-based solver.

- Our method is not without limitations.



- As in rendering, coefficients with large spatial variation can lead to increased variance.
- Adapting further techniques from volume rendering such as local coefficient bounds, low-variance VRE estimators, and adaptive weight windows should help address this issue.

Future: support for important features

Neumann & Robin boundary conditions

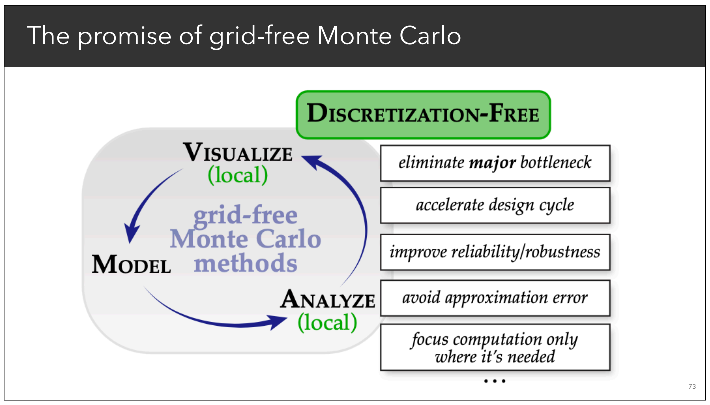
Anisotropic diffusion coefficients

Non-linear PDEs

High performance distance queries

Differentiable implementation

- More broadly, the WoS framework still lacks support for many basic features of schemes like FEM, such as Neumann boundary conditions and anisotropic diffusion coefficients.
- That said, this framework is still a very interesting [fairly new] way to solve PDEs, with deep-but-unexplored connections to rendering.



- Moreover, since Monte Carlo methods are free from the bonds of spatial discretization, they open the door to new classes of PDE solvers that are robust to bad geometry, scalable to extremely large scenes, and progressive in their solution evaluation.

- Thank you.

BACKUP

I don't know…