Physically-Based Animation and PDEs

Computer Graphics
CMU 15-462/15-662

Last time: Optimization

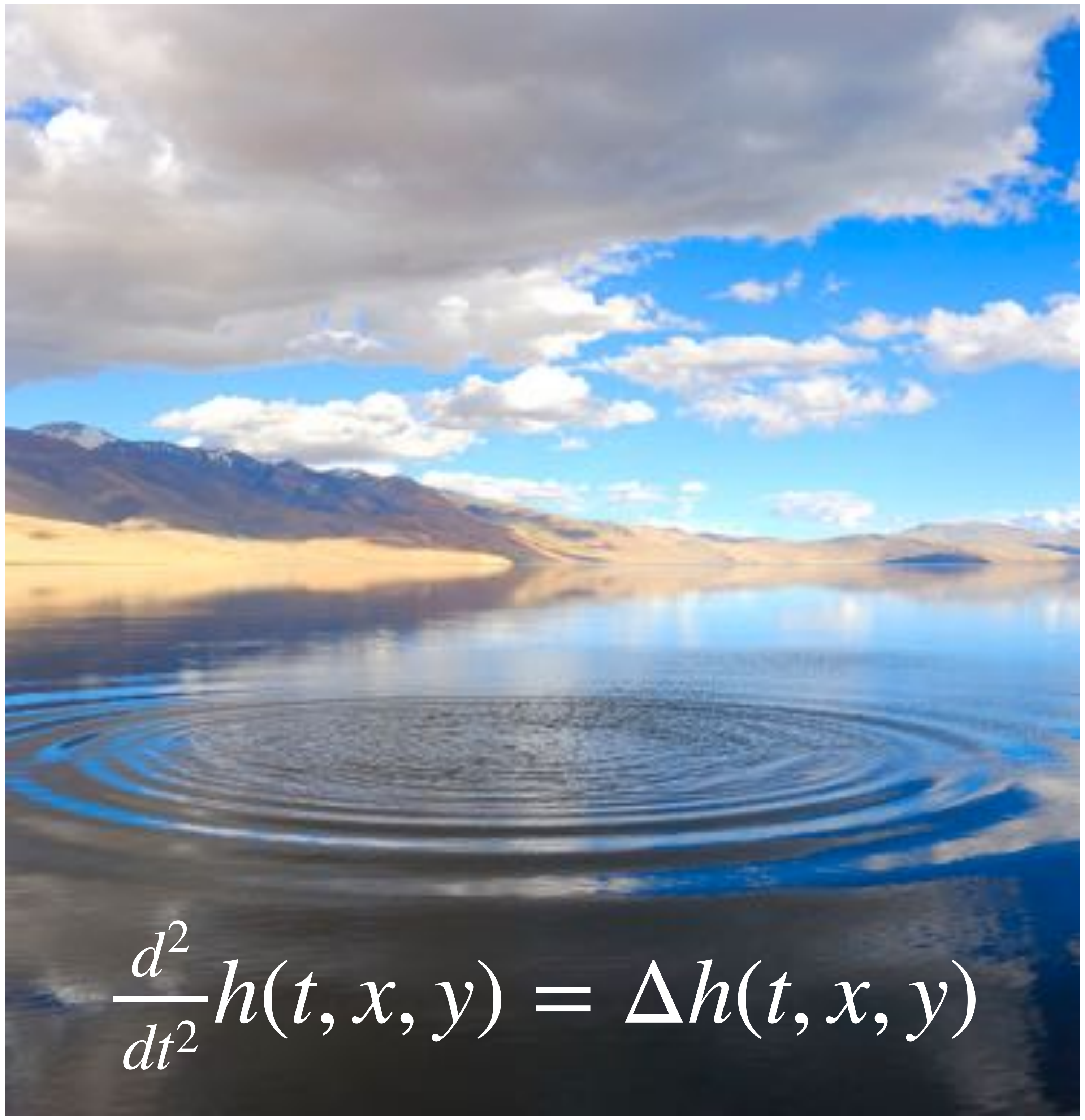
- **Modern graphics uses optimization!**
- **Many complex criteria/constraints**
- **Basic technique: numerical descent**
  - **pick initial guess**
  - **ski downhill**
  - **keep fingers crossed!**
- **Gradient descent important example of ordinary differential equation (ODE)**
- **Today: return to differential equations**
  - **saw ODEs—derivatives in time**
  - **now PDEs—also have derivatives in space**
  - **describe many natural phenomena (water, smoke, cloth, ...)**
  - **recent revolution in CG/visual effects**

Partial Differential Equations (PDEs)

- ODE: Implicitly describe function in terms of its <u>time derivatives</u>
- PDE: Also include <u>spatial derivatives</u> in implicit description
- Like any implicit description, have to solve for actual function

ODE—rock flies through air

PDE—rock lands in pond



To make a long story short...

- **Solving ODE looks like "add a little velocity each time"**

$$q_{k+1} = q_k + \tau f(q)$$

- **Solving a PDE looks like "take weighted combination of neighbors to get velocity (...and add a little velocity each time)"**

$$\begin{array}{|c|c|c|} \hline & 1 & \\ \hline 1 & -4 & 1 \\ \hline & 1 & \\ \hline \end{array}$$

$$f(q)$$

$$q_{k+1} = q_k + \tau f(q)$$

...obviously there is a lot more to say here!

Solving a PDE in Code

Don’t be intimidated—very simple code can give rise to beautiful behavior!

```
void simulateWaves2D() {
   const int N = 128; // grid size
   double u[N][N]; // height
   double v[N][N]; // velocity (time derivative of height)
   const double tau = 0.2; // time step size
   const double alpha = 0.985; // damping factor

   for( int frame = 0; true; frame++ ) {  // loop forever
      // drop random "stones"
      if( frame % 100 == 0 ) u[rand()%N][rand()%N] = -1;
      // update velocity
      for( int i = 0; i < N; i++ )
      for( int j = 0; j < N; j++ ) {
         int i0 = (i + N-1) % N; // left
         int i1 = (i + N+1) % N; // right
         int j0 = (j + N-1) % N; // down
         int j1 = (j + N+1) % N; // up
         v[i][j] += tau * (u[i0][j] + u[i1][j] + u[i][j0] + u[i][j1] - 4*u[i][j])
         v[i][j] *= alpha; // damping
      }
      // update height
      for( int i = 0; i < N; i++ )
      for( int j = 0; j < N; j++ ) {
         u[i][j] += tau * v[i][j];
      }
      display( u );
   }
}
```

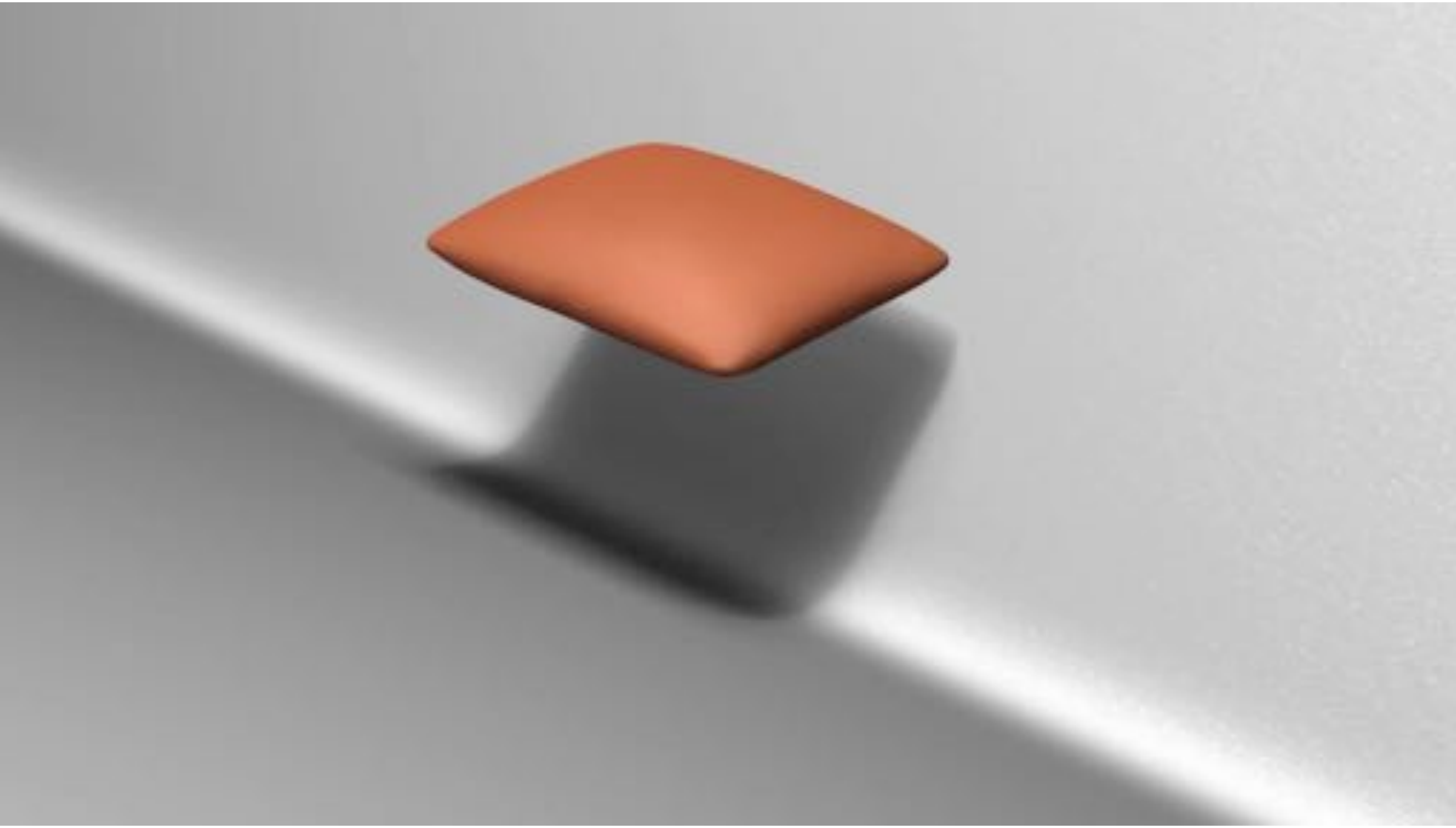
Liquid Simulation in Graphics

Losasso, F., Shinar, T. Selle, A. and Fedkiw, R., "Multiple Interacting Liquids"

Smoke Simulation in Graphics

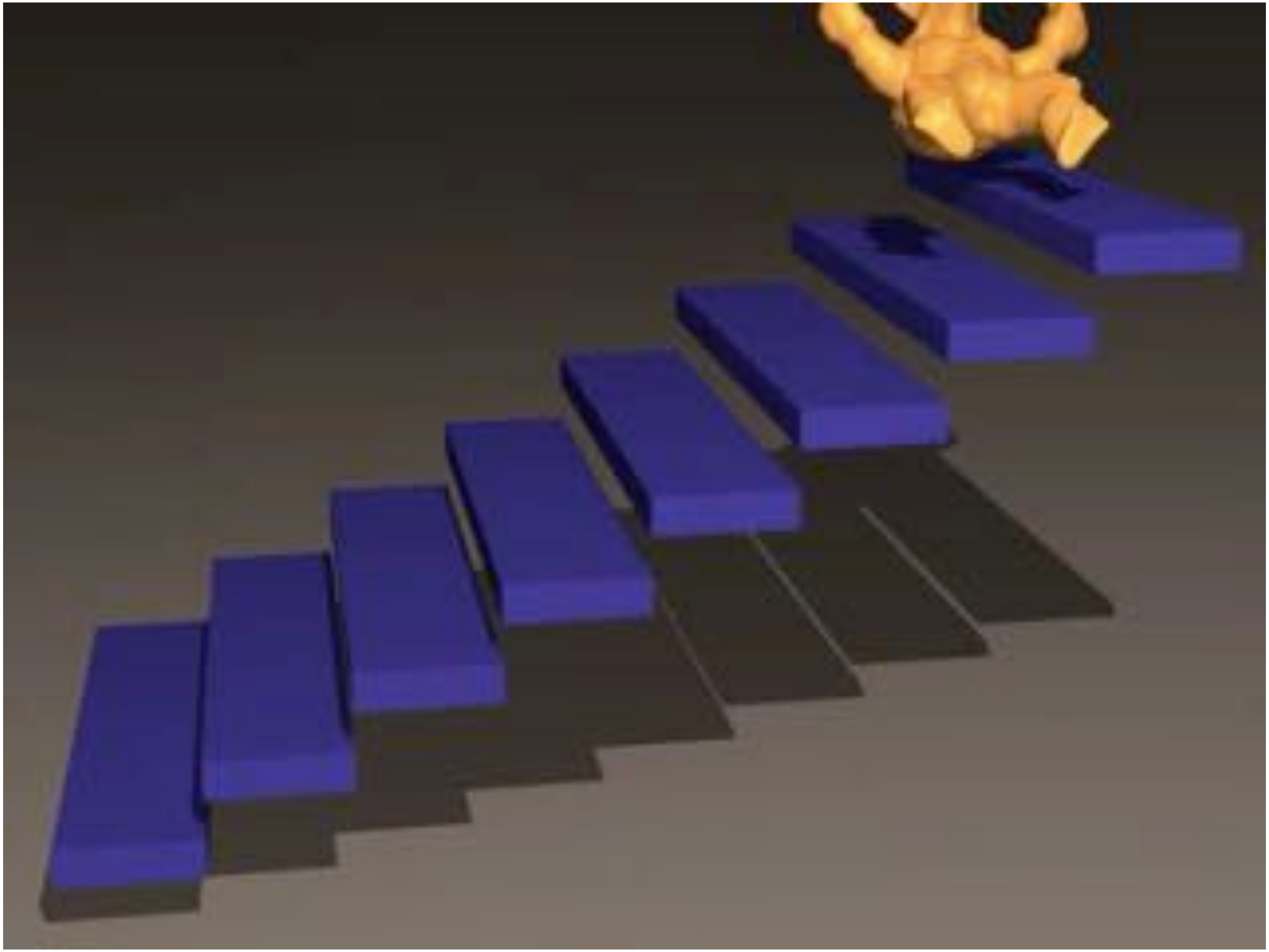
S. Weißmann, U. Pinkall. “Filament-based smoke with vortex shedding and variational reconnection”

Cloth Simulation in Graphics



Zhili Chen, Renguo Feng and Huamin Wang, “Modeling friction and air effects between cloth and deformable bodies”

Elasticity in Graphics

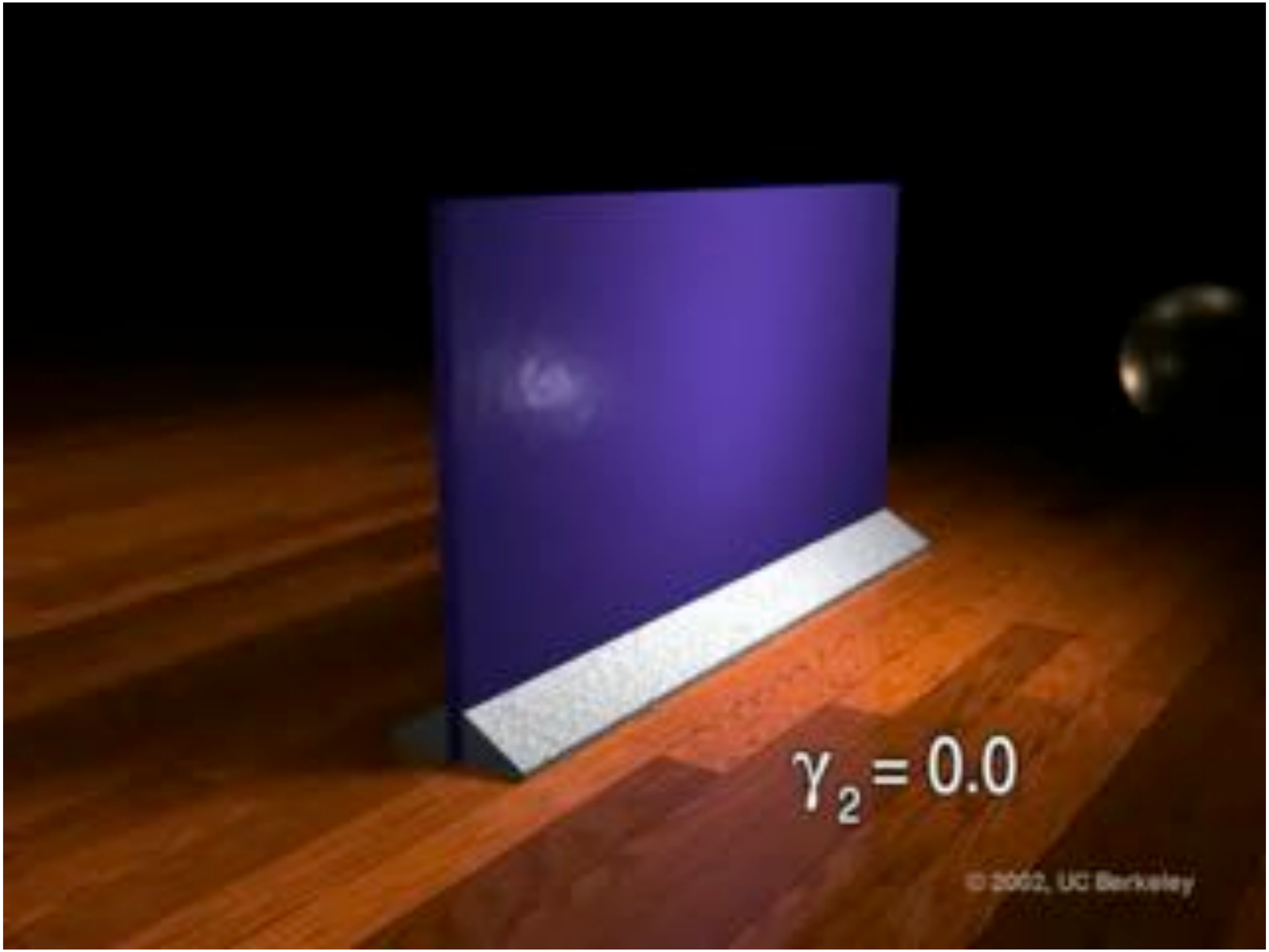


Irving, G., Schroeder, C. and Fedkiw, R., "Volume Conserving Finite Element Simulation of Deformable Models"

Hair Simulation in Graphics

Danny M. Kaufman, Rasmus Tamstorf, Breannan Smith, Jean-Marie Aubry, Eitan Grinspun, "Adaptive Nonlinearity for Collisions in Complex Rod Assemblies"

Fracture in Graphics



James F. O'Brien, Adam Bargteil, Jessica Hodgins, "Graphical Modeling and Animation of Ductile Fracture"

Viscoelasticity in Graphics

Chris Wojtan, Greg Turk, “Fast Viscoelastic Behavior with Thin Features”

Snow Simulation in Graphics

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, Andrew Selle, "A Material Point Method For Snow Simulation"

Definition of a PDE

- **Want to solve for a function of time and space**

$$u(\underset{\text{time}}{t}, \underset{\text{space}}{x})$$

- **Function given implicitly in terms of derivatives:**

$$\dot{u}, \ddot{u}, \frac{d^3u}{dt^3}, \frac{d^4u}{dt^4}, \ldots$$

any combination of time derivatives

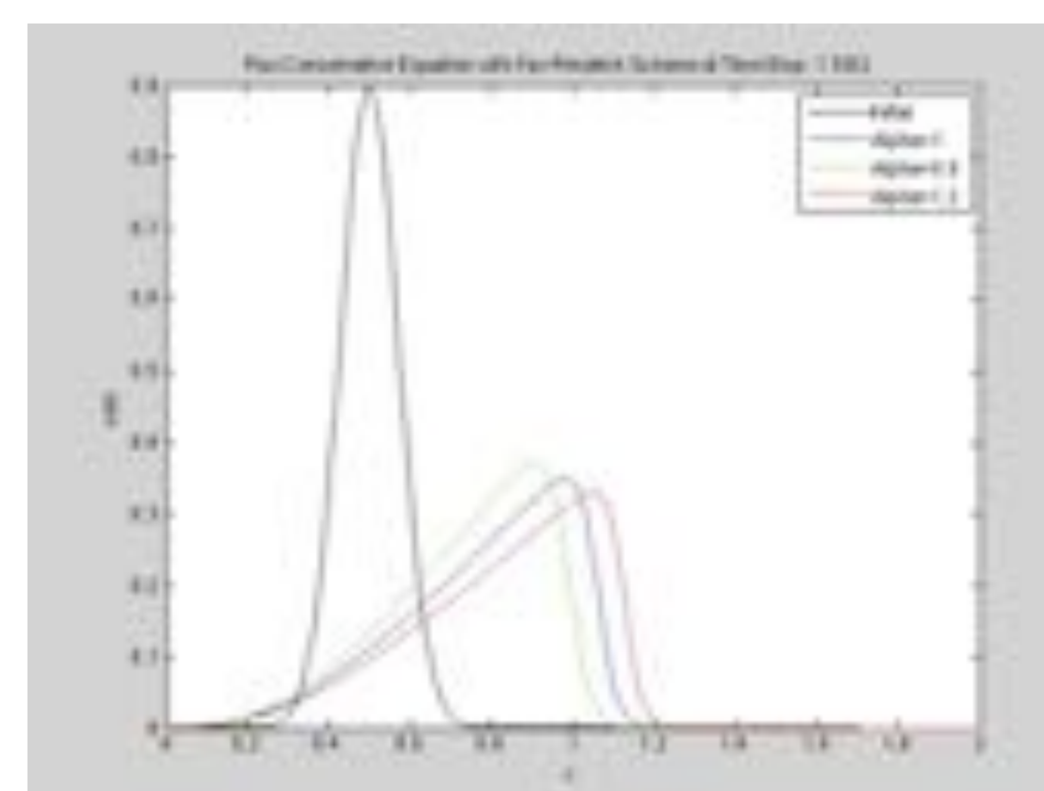
$$\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \frac{\partial^{m+n} u}{\partial x_i^m \partial x_j^n}, \ldots$$

plus any combination of space derivatives

- **Example:**

$$\underset{\frac{du}{dt}}{\dot{u}} + u\overset{\frac{\partial u}{\partial x}}{u'} = \alpha \overset{\frac{\partial^2 u}{\partial x^2}}{u''}$$

(Burgers' equation)



Anatomy of a PDE

- **Linear vs. nonlinear: how are derivatives combined?**

nonlinear!

$$\dot{u} + uu' = au''$$

(Burgers' equation)

$$\dot{u} = au''$$

(diffusion equation)

- **Order: how many derivatives in space & time?**

1st order in time

2nd order in space

$$\dot{u} + uu' = au''$$

(Burgers' equation)

2nd order in time

2nd order in space

$$\ddot{u} = au''$$

(wave equation)

Rule of thumb: nonlinear / higher order $\Rightarrow$ HARDER TO SOLVE!

Model Equations

- Fundamental behavior of many important PDEs is well-captured by three model linear equations:

Solve numerically?

"Laplacian" (more later!)

LAPLACE EQUATION ("ELLIPTIC") $\Delta u = 0$

"what's the smoothest function interpolating the given boundary data"

HEAT EQUATION ("PARABOLIC") $\dot{u} = \Delta u$

"how does an initial distribution of heat spread out over time?"

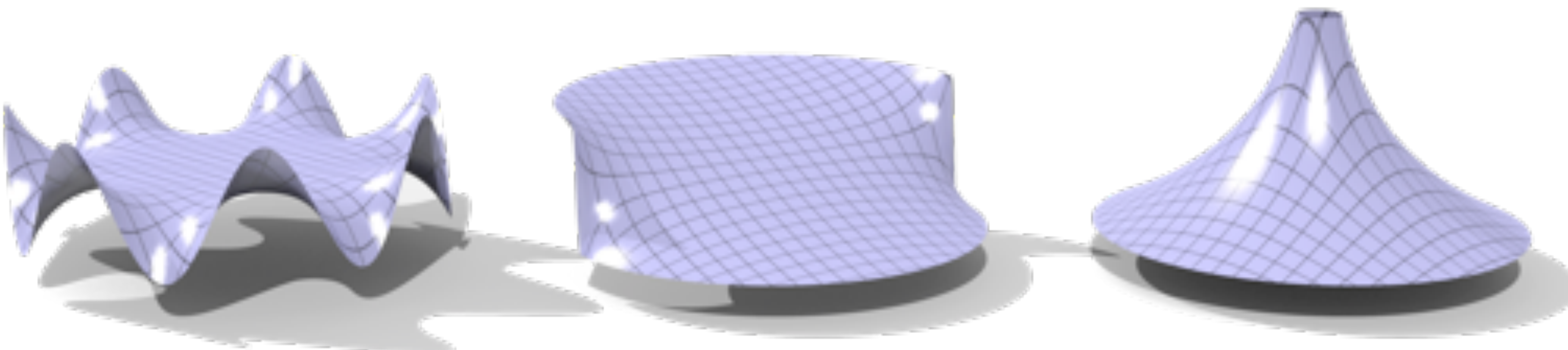
WAVE EQUATION ("HYPERBOLIC") $\ddot{u} = \Delta u$

"if you throw a rock into a pond, how does the wavefront evolve over time?"

[NONLINEAR + HYPERBOLIC + HIGH-ORDER]

Elliptic PDEs / Laplace Equation

- "What's the smoothest function interpolating the given boundary data?"

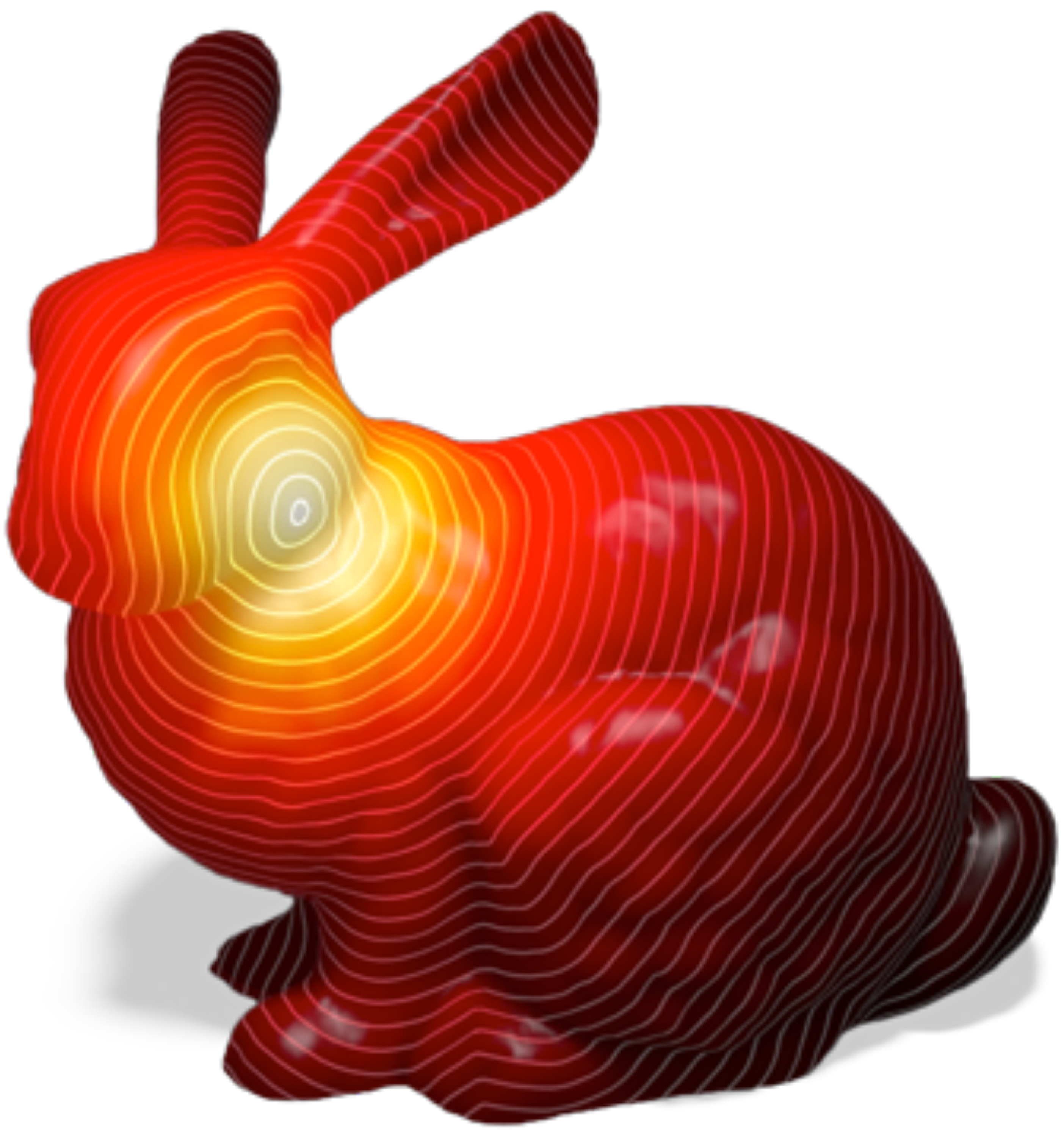


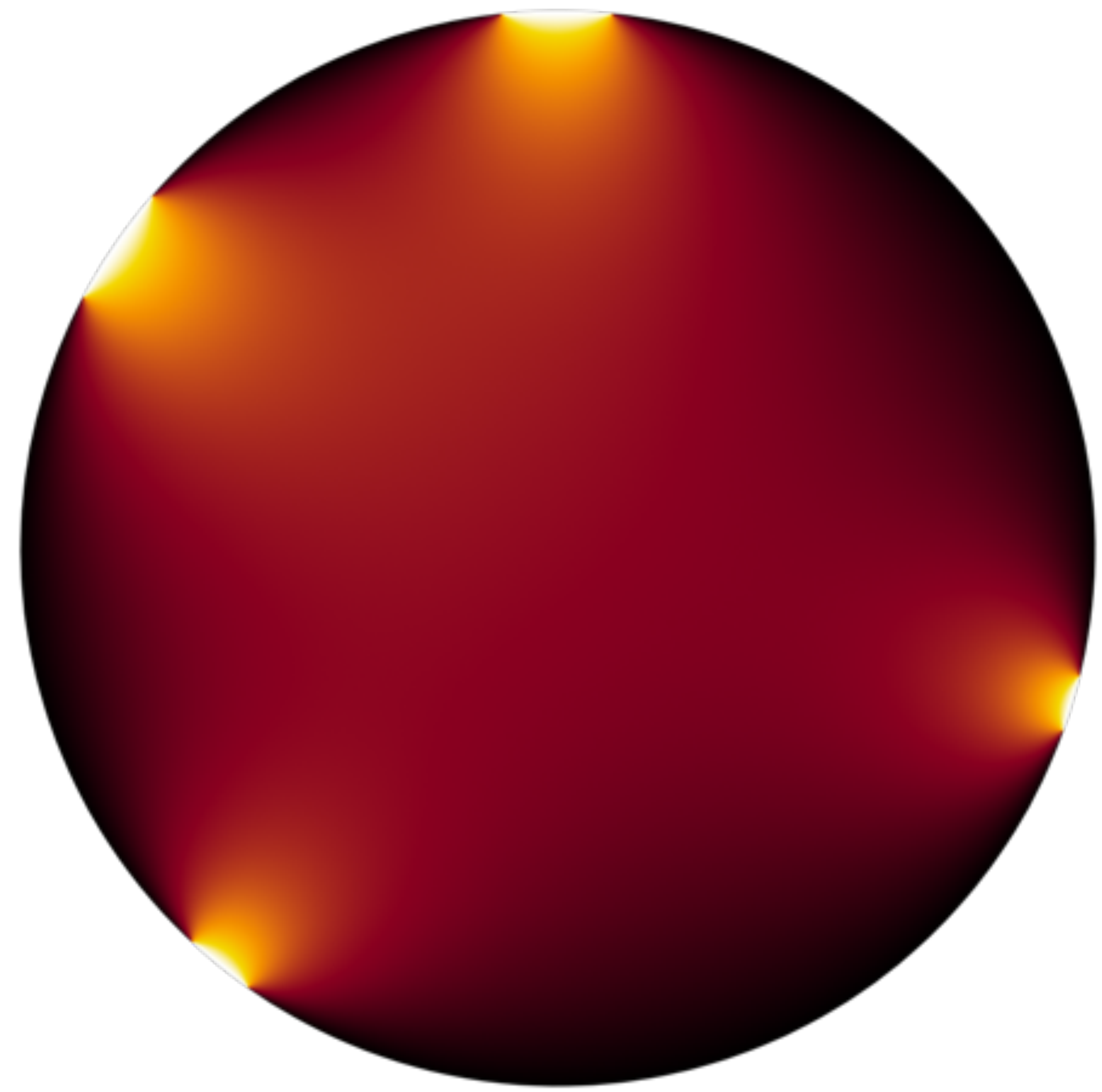
- Conceptually: each value is at the average of its "neighbors"
- Roughly speaking, why is it easier to solve?
- Very robust to errors: just keep averaging with neighbors!

Image from Solomon, Crane, Vouga, "Laplace-Beltrami: The Swiss Army Knife of Geometry Processing"

Parabolic PDEs / Heat Equation

- "How does an initial distribution of heat spread out over time?"





- After a long time, solution is same as Laplace equation!
- Models damping / viscosity in many physical systems

Hyperbolic PDEs / Wave Equation

- "If you throw a rock into a pond, how does the wavefront evolve over time?"

- Errors made at the beginning will persist for a long time! (hard)

PDEs give an implicit description of solution.

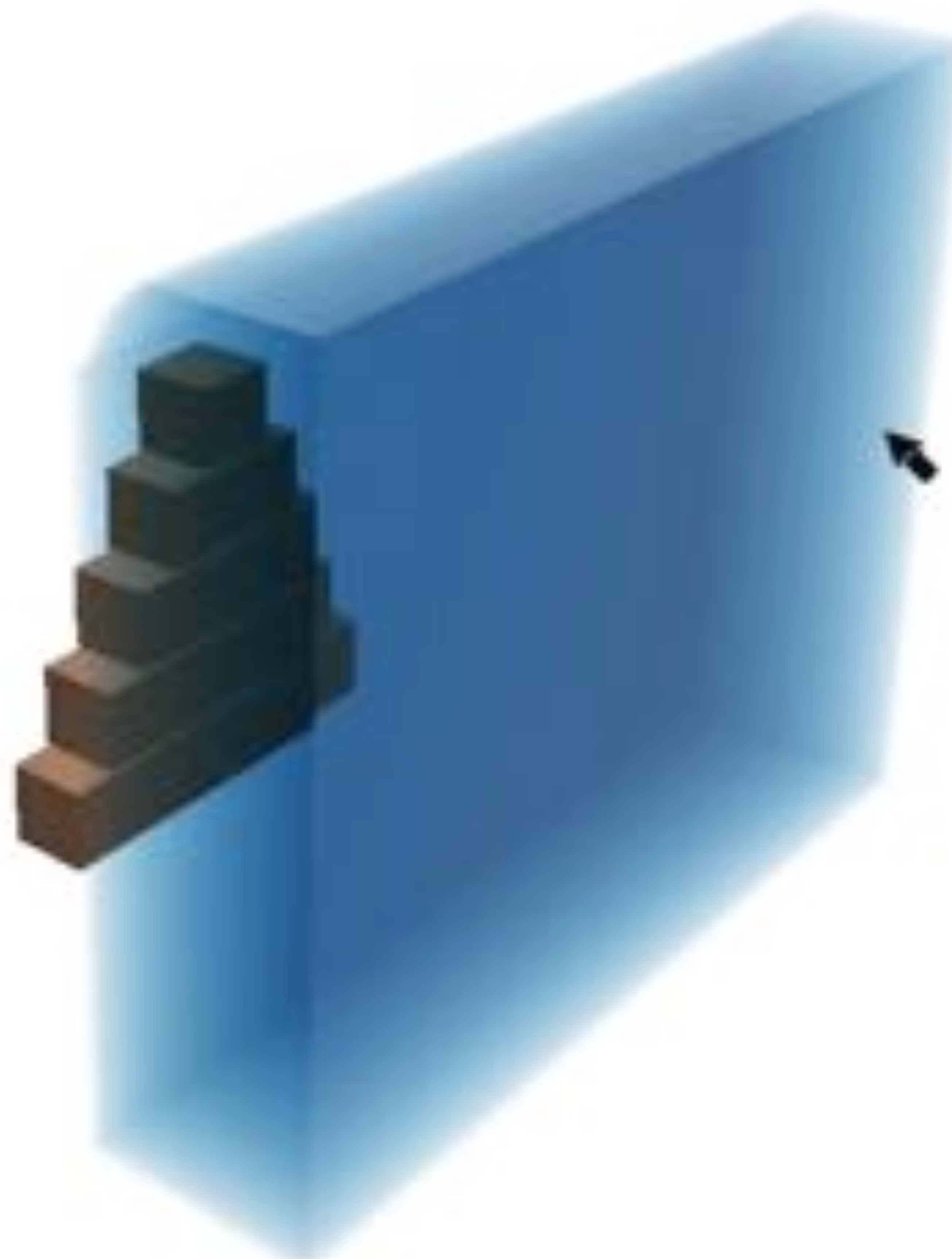
How do we compute solutions explicitly?

Numerical Solution of PDEs—Overview

- Like ODEs, most PDEs are difficult/impossible to solve analytically—especially if we want to incorporate data!
- Must instead use numerical time integration
- Basic strategy:
  - –pick a <u>time</u> discretization (forward Euler, backward Euler...)
  - –pick a <u>spatial</u> discretization (TODAY)
  - –as with ODEs, perform time-stepping to advance solution
- Historically, very expensive—only for "hero shots" in movies
- Computers are ever faster...
- More & more use of PDEs
  - games, interactive tools, ...

Real Time PDE-Based Simulation (Fire)

Real Time PDE-Based Simulation (Water)

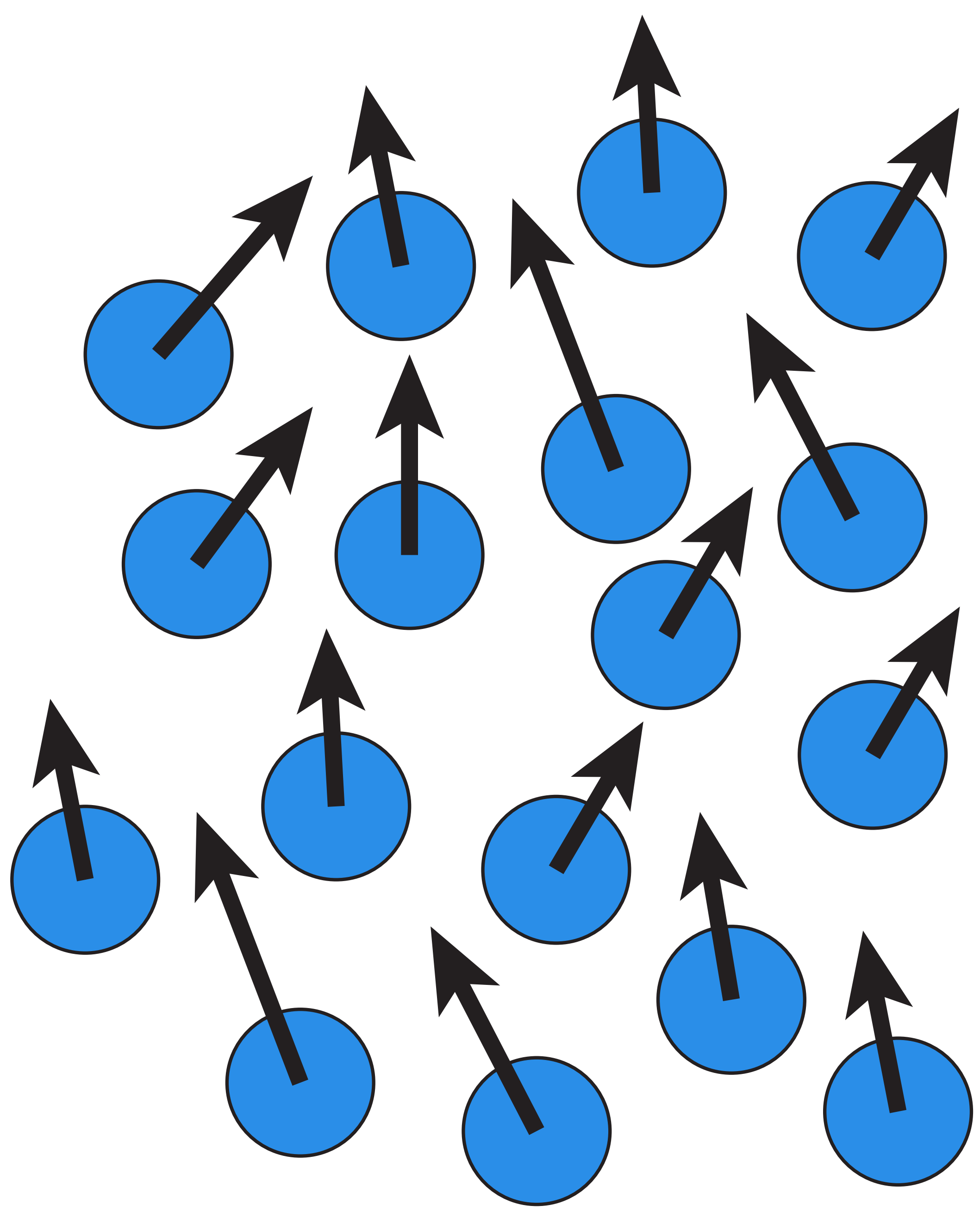


Nuttapong Chentanez, Matthias Müller, “Real-time Eulerian water simulation using a restricted tall cell grid” (2011)

Lagrangian vs. Eulerian

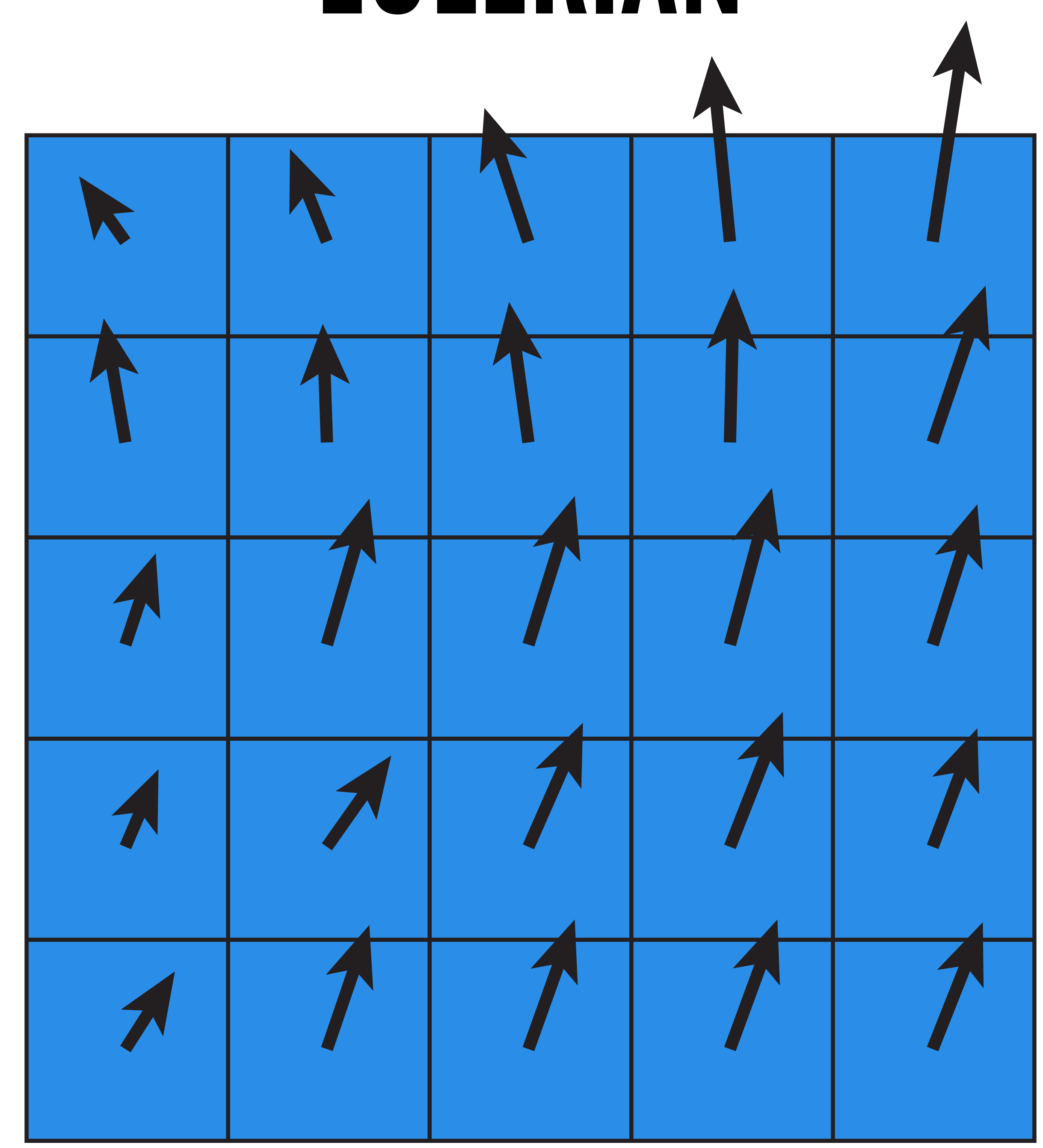
- Two basic ways to discretize space: Lagrangian & Eulerian
- E.g., suppose we want to encode the motion of a fluid

LAGRANGIAN



track position & velocity
of moving particles

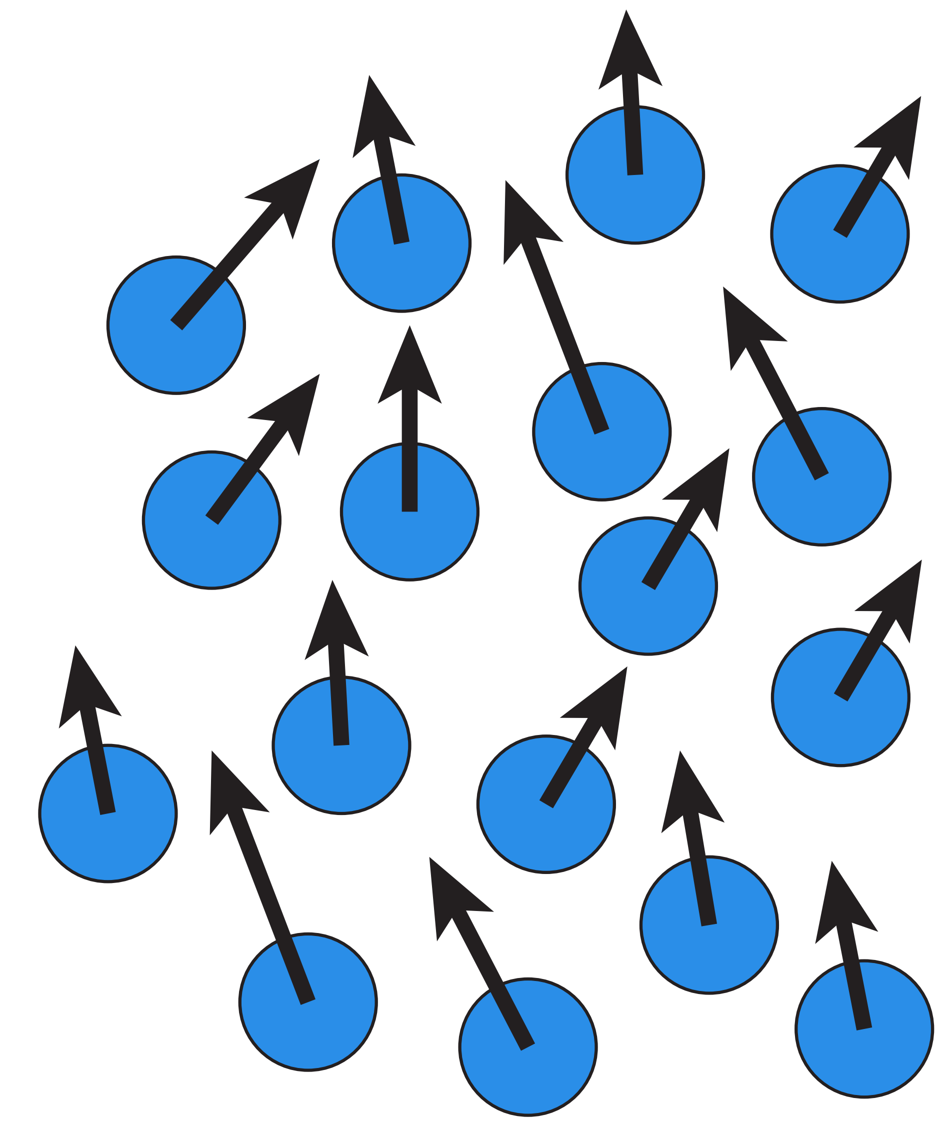
EULERIAN

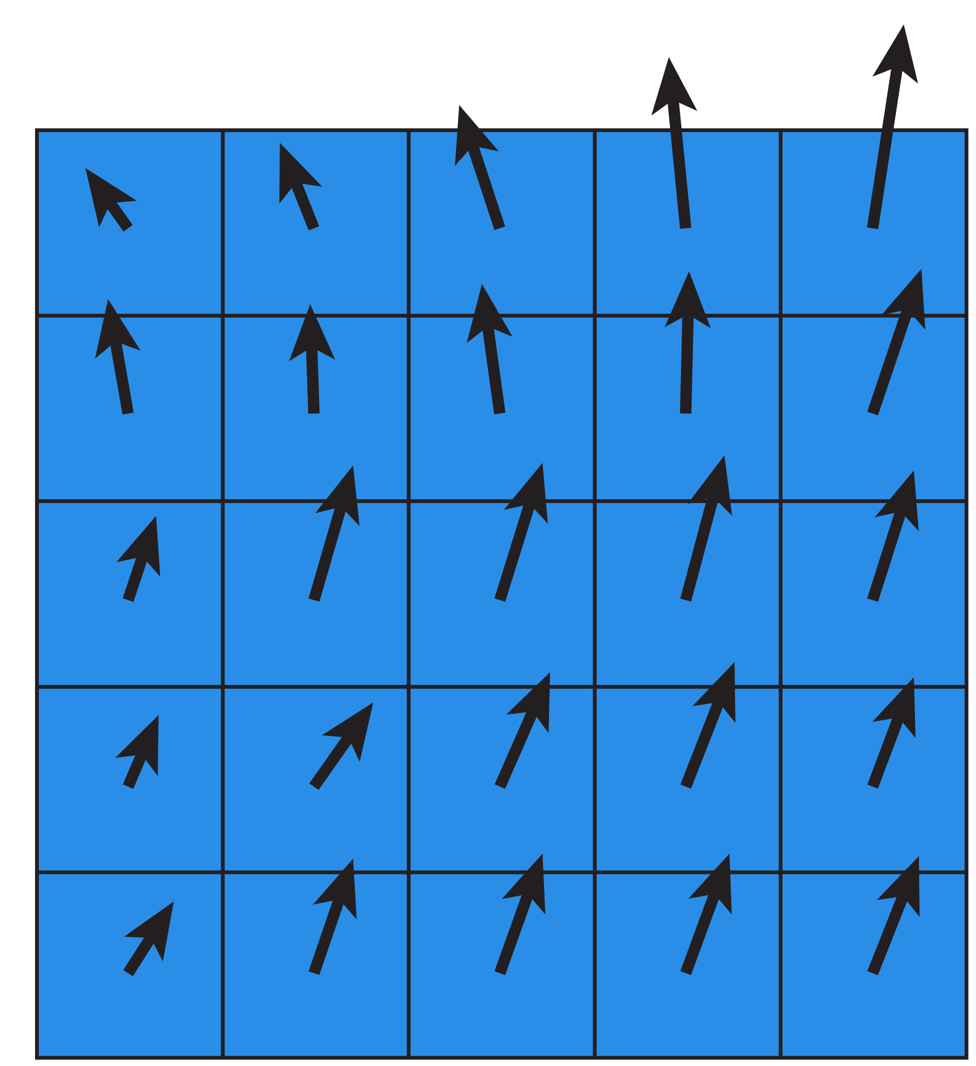


track velocity (or flux)
at fixed grid locations

Lagrangian vs. Eulerian—Trade-Offs

- **Lagrangian**
  - conceptually easy (like polygon soup!)
  - resolution/domain not limited by grid
  - good particle distribution can be tough
  - finding neighbors can be expensive
- **Eulerian**
  - fast, regular computation
  - easy to represent, e.g., smooth surfaces
  - simulation "trapped" in grid
  - grid causes "numerical diffusion" (blur)
  - need to understand PDEs (but you will!)

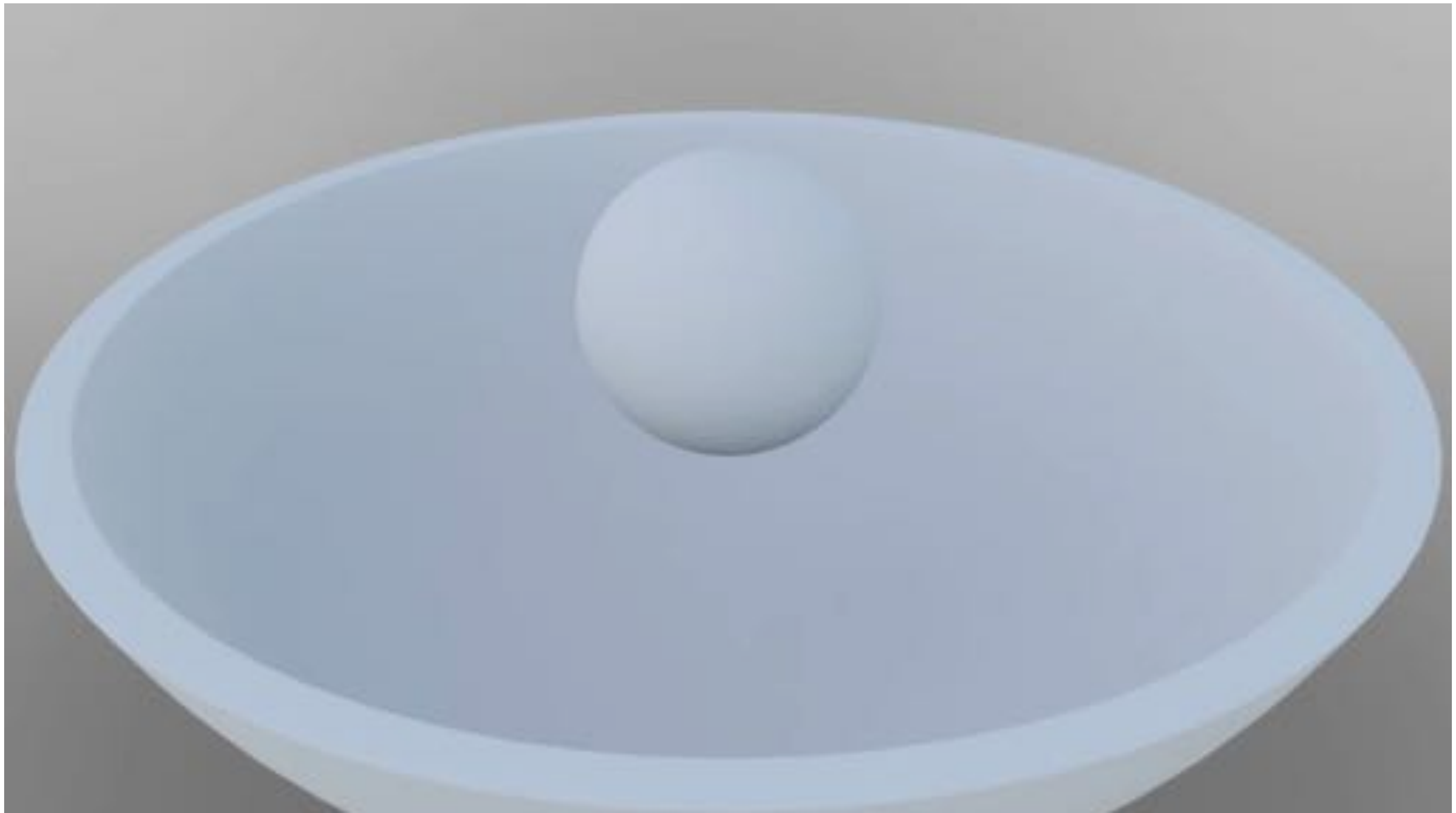




Mixing Lagrangian & Eulerian

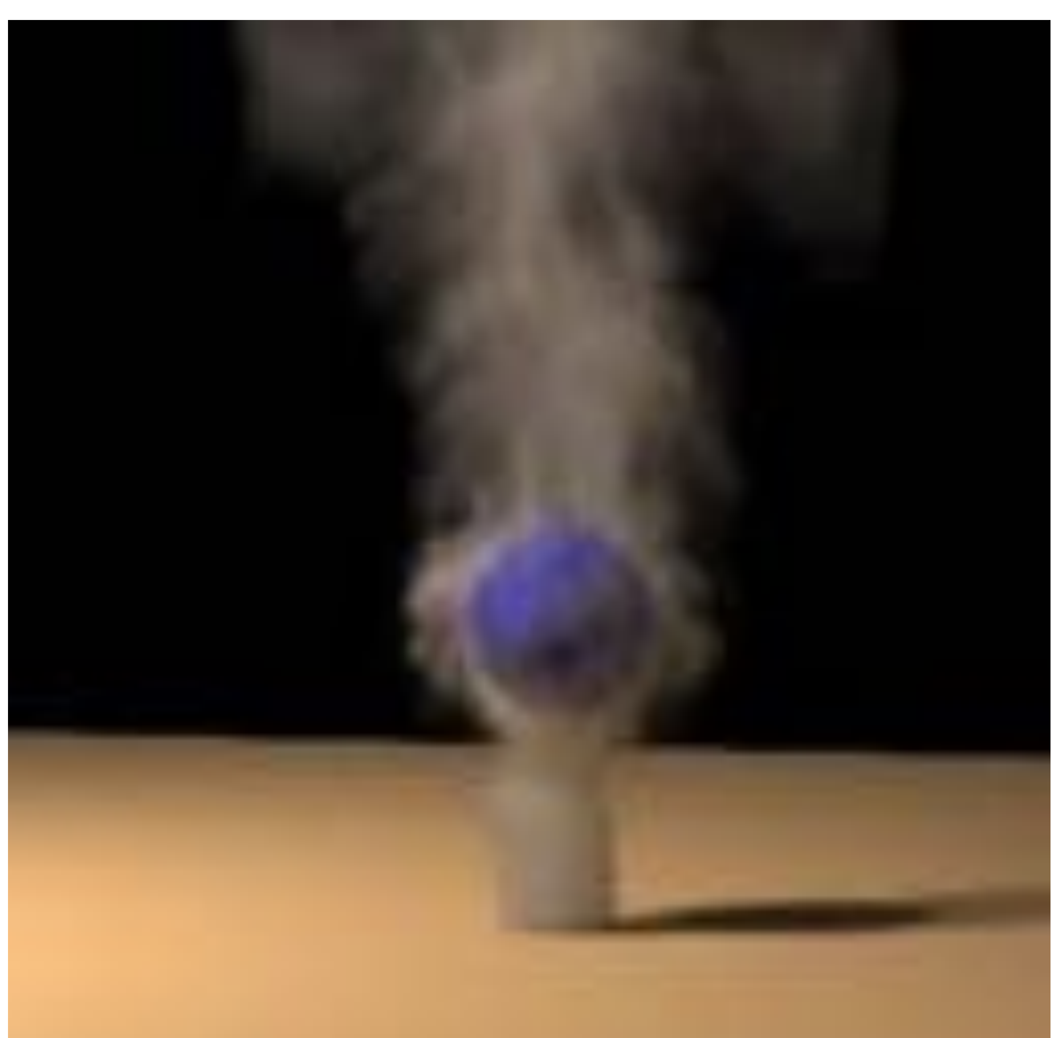
- **Of course, no reason you have to choose just one!**
- **Many modern methods mix Lagrangian & Eulerian:**
  - **PIC/FLIP, particle level sets, mesh-based surface tracking, Voronoi-based, arbitrary Lagrangian-Eulerian (ALE), ...**
- **Pick the right tool for the job!**

Maya Bifrost



Aside: Which Quantity Do We Solve For?

- Many PDEs have mathematically equivalent formulations in terms of different quantities
- E.g., incompressible fluids:
  - velocity—how fast is each particle moving?
  - vorticity—how fast is fluid "spinning" at each point?
- Computationally, can make a big difference
- Pick the right tool for the job!



Ok, but we're getting way ahead of ourselves. How do we solve easy PDEs?

Numerical PDEs—Basic Strategy

Richard Courant

- Pick PDE formulation
  - Which quantity do we want to solve for?
  - E.g., velocity or vorticity?
- Pick spatial discretization
  - How do we approximate derivatives in space?
- Pick time discretization
  - How do we approximate derivatives in time?
  - When do we evaluate forces?
  - Forward Euler, backward Euler, symplectic Euler, ...
- Finally, we have an update rule
- Repeatedly solve to generate an animation

The Laplace Operator

- All of our model equations used the Laplace operator
- Different conventions for symbol:

$$\Delta$$ same symbol used for "change"

$$\nabla^2$$ same symbol used for Hessian!

- Unbelievably important object showing up everywhere across physics, geometry, signal processing, ...
- Ok, but what does it mean?
- Differential operator: eats a function, spits out its "2nd derivative"
- What does that mean for a function $u : \mathbb{R}^n \to \mathbb{R}$?
  - –divergence of gradient
  - –sum of second derivatives
  - –deviation from local average
  - –...

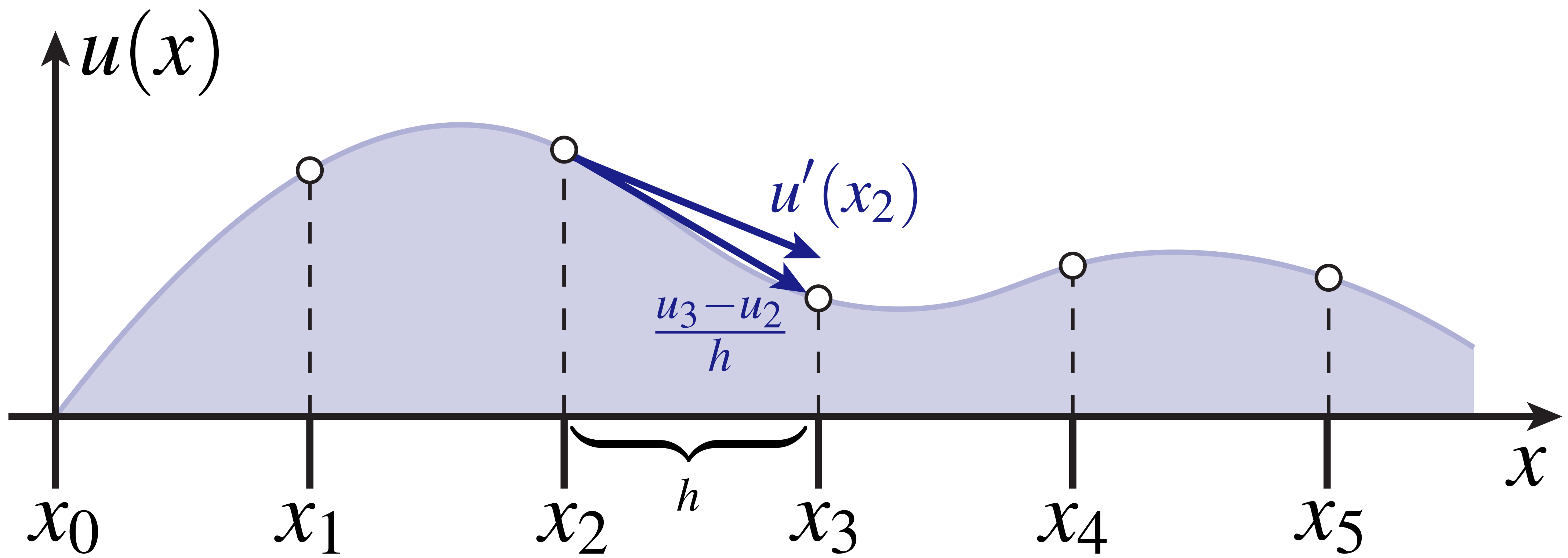
$$\Delta u = \underbrace{\nabla}_{\text{div}} \cdot \underbrace{\nabla}_{\text{grad}} u$$

$$\Delta u = \frac{\partial u^2}{\partial x_1^2} + \cdots + \frac{\partial u^2}{\partial x_n^2}$$

For more intuition about the Laplacian: https://youtu.be/oEq9ROl9Umk

Discretizing the First Derivative

- **To solve any PDE, need to approximate spatial derivatives (e.g., Laplacian)**
- **Suppose we know a function $u(x)$ only at regular intervals h**



- **Q: How can we approximate the <u>first</u> derivative of u?**
- **A: Recall definition of a derivative in terms of limits:**

$$u'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}$$

- **Can hence get an approximation using known values:**

$$u'(x_i) \approx \frac{u_{i+1} - u_i}{h}$$

- **Approximation gets better for finer grid (smaller h)**

Discretizing the Second Derivative

- **Q: How can we get an approximation of the second derivative?**
- **A: One idea*: approximate the first derivative of the approximate first derivative!**

$$u''(x_i) \approx \frac{u'_i - u'_{i-1}}{h} \approx \frac{\left(\frac{u_{i+1}-u_i}{h}\right) - \left(\frac{u_i - u_{i-1}}{h}\right)}{h} =$$

$$\boxed{\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}}$$

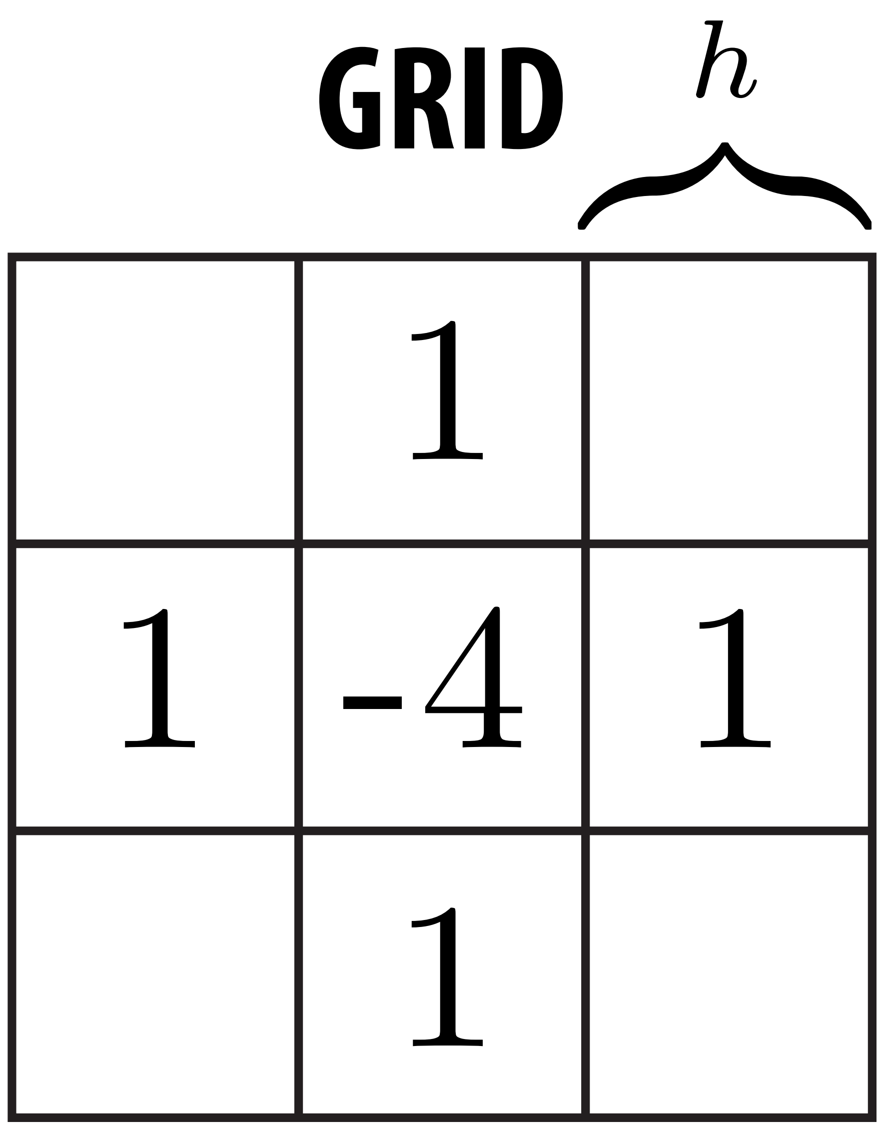
- **In general, this approach of approximating derivatives with differences is the "finite difference" approach to PDEs**
- **Not the only way! But works well on regular grids.**

*Can show this is also a reasonable thing to do, using Taylor series

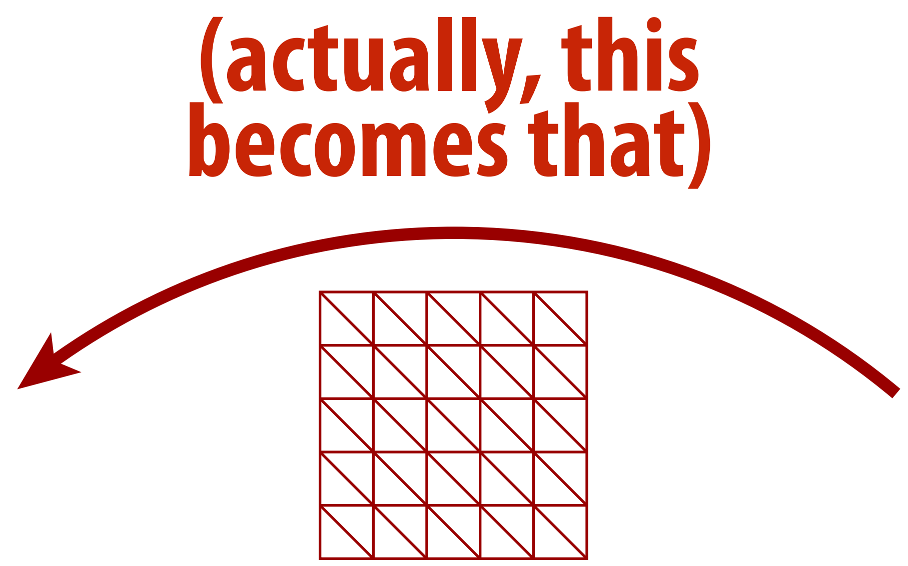
Discretizing the Laplacian

- How do we approximate the Laplacian?
- Depends on discretization (Eulerian, Lagrangian, grid, mesh, ...)
- Two extremely common ways in graphics:

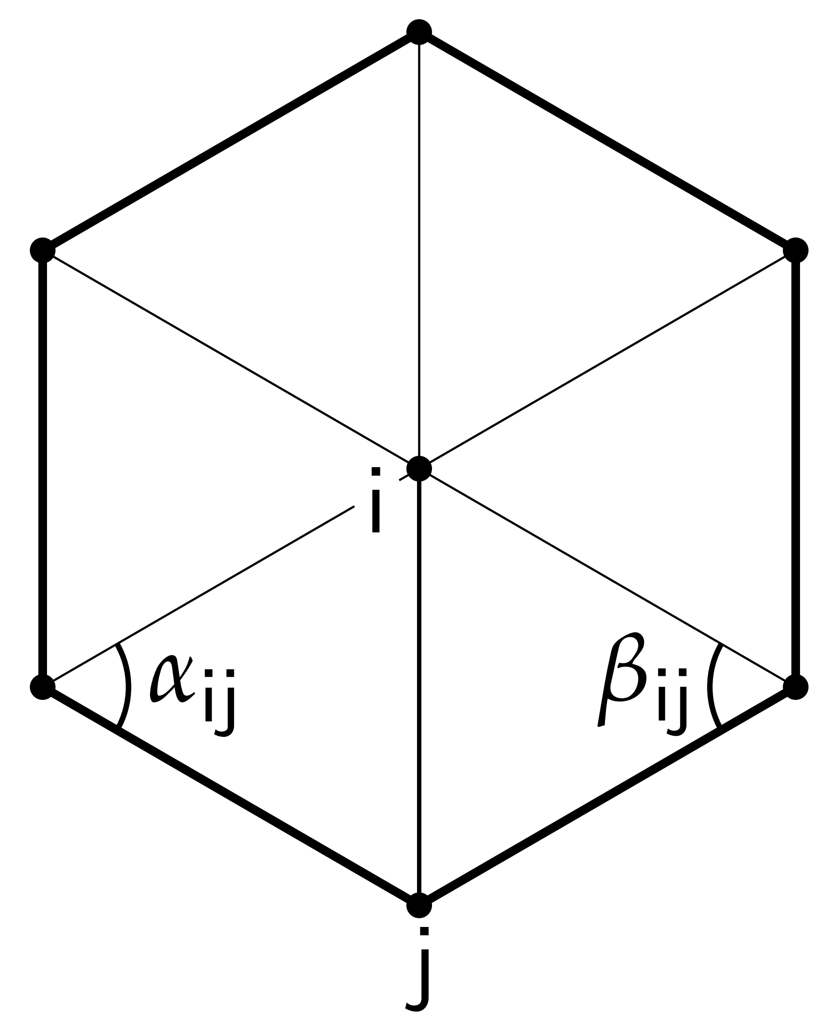
GRID



$$\frac{4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}}{h^2}$$



TRIANGLE MESH



$$\frac{1}{2}\sum_j (\cot\alpha_{ij} + \cot\beta_{ij})(u_j - u_i)$$

- Also not too hard on point clouds, polygon meshes, ...

Numerically Solving the Laplace Equation

- Want to solve $\Delta u = 0$
- Plug in one of our discretizations, e.g.,

	$u_{i,j+1}$	
$u_{i-1,j}$	$u_{i,j}$	$u_{i+1,j}$
	$u_{i,j-1}$	

$$\frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2} = 0$$

$$\iff u_{i,j} = \frac{1}{4}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right)$$

- If u is a solution, then each value must be the average of the neighboring values (u is a "harmonic function")
- How do we solve this?
- One idea: keep averaging with neighbors! ("Jacobi method")
- Correct, but slow. Much better to use modern linear solver

Aside: PDEs and Linear Equations

- **How can we turn our Laplace equation into a linear solve?**
- **Have a bunch of equations of the form**

$$4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = 0$$

- **On a 4x4 grid, assign each cell $u_{i,j}$ a unique index 1, …, 16**

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- **Can then write equations as a 16x16 matrix equation***

$$\begin{bmatrix}
-4 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & -4 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \\ u_{15} \\ u_{16} \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- **Compute solution by calling <u>sparse</u> linear solver (SuiteSparse, Eigen, …)**
- **Q: By the way, what's wrong with our problem setup here? :-)**

*assuming neighbors wrap around left/right and top/bottom

Boundary Conditions for Discrete Laplace

- **What values do we use to compute averages near the boundary?**

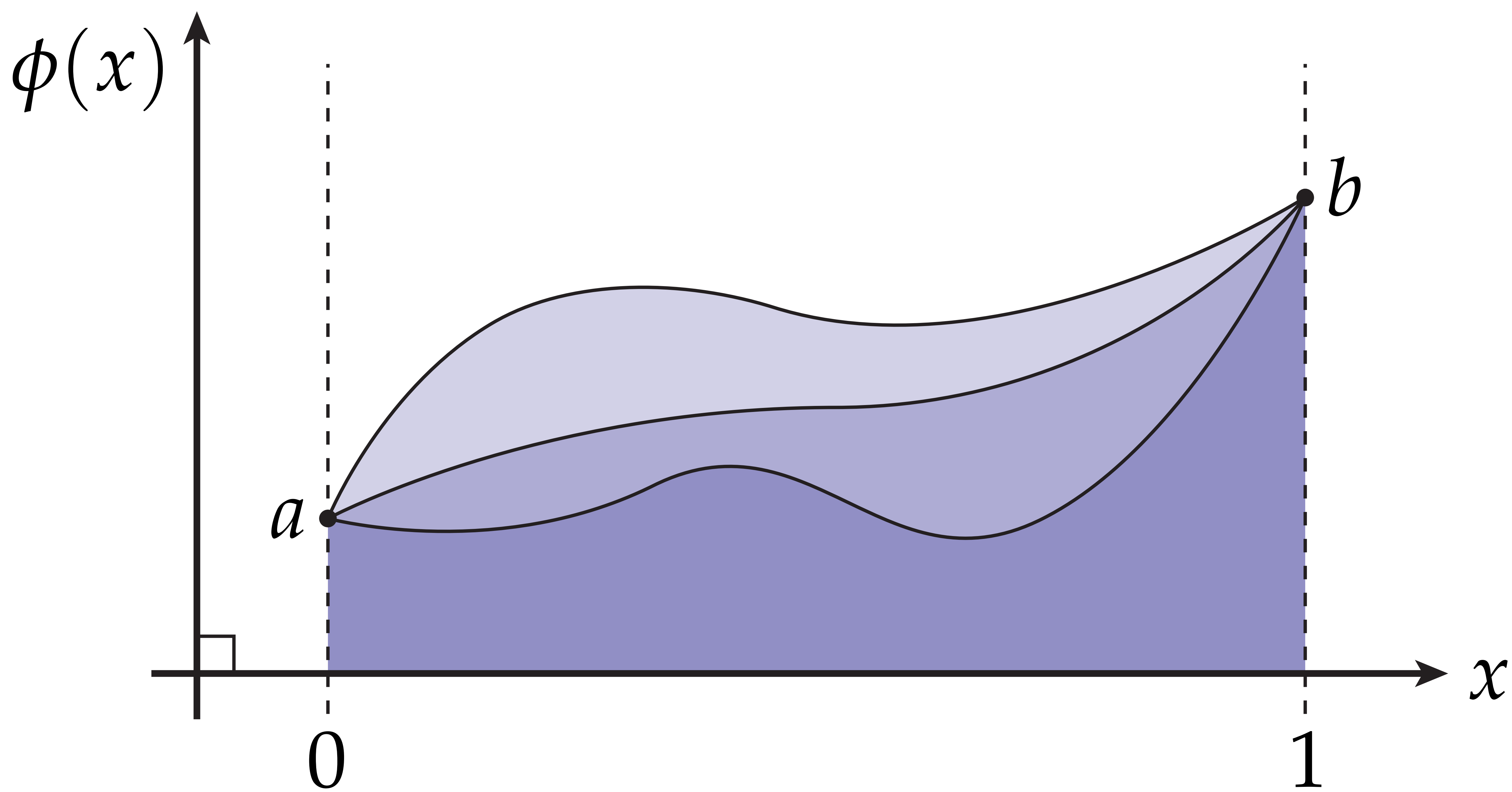
	c	
$?$	a	b
	e	

$$a = \frac{1}{4}(b + c + ? + e)$$

- **A: We get to choose—this is the data we want to interpolate!**
- **Two basic boundary conditions:**
  1. **Dirichlet—boundary data always set to fixed values**
  2. **Neumann—specify derivative (difference) across boundary**
- **Also mixed (Robin) boundary conditions (and more, in general)**

Dirichlet Boundary Conditions

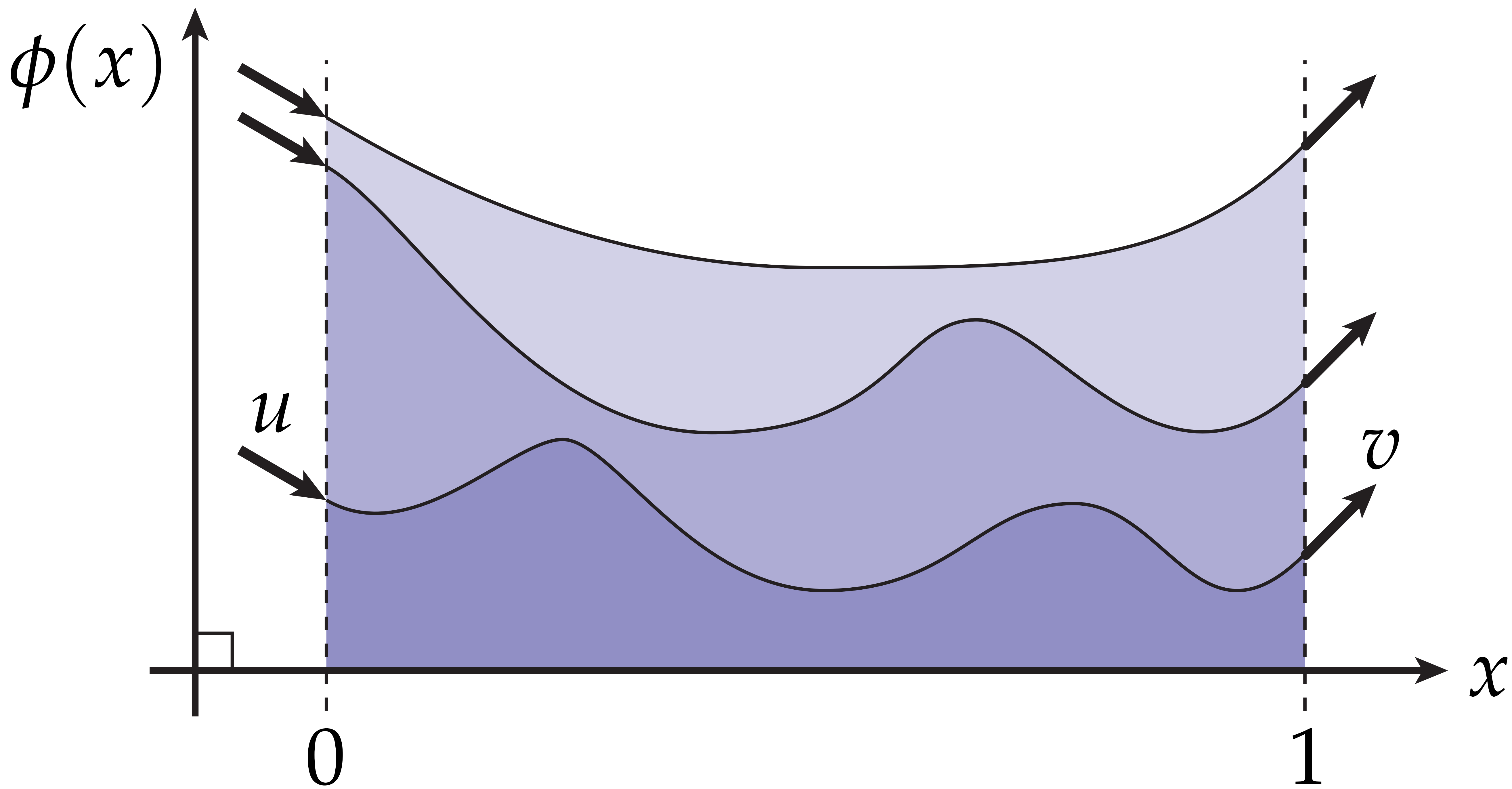
- Let's go back to smooth setting, function on real line
- Dirichlet means "prescribe values"
- E.g., $\phi(0) = a, \phi(1) = b$



- Many possible functions "in between"!

Neumann Boundary Conditions

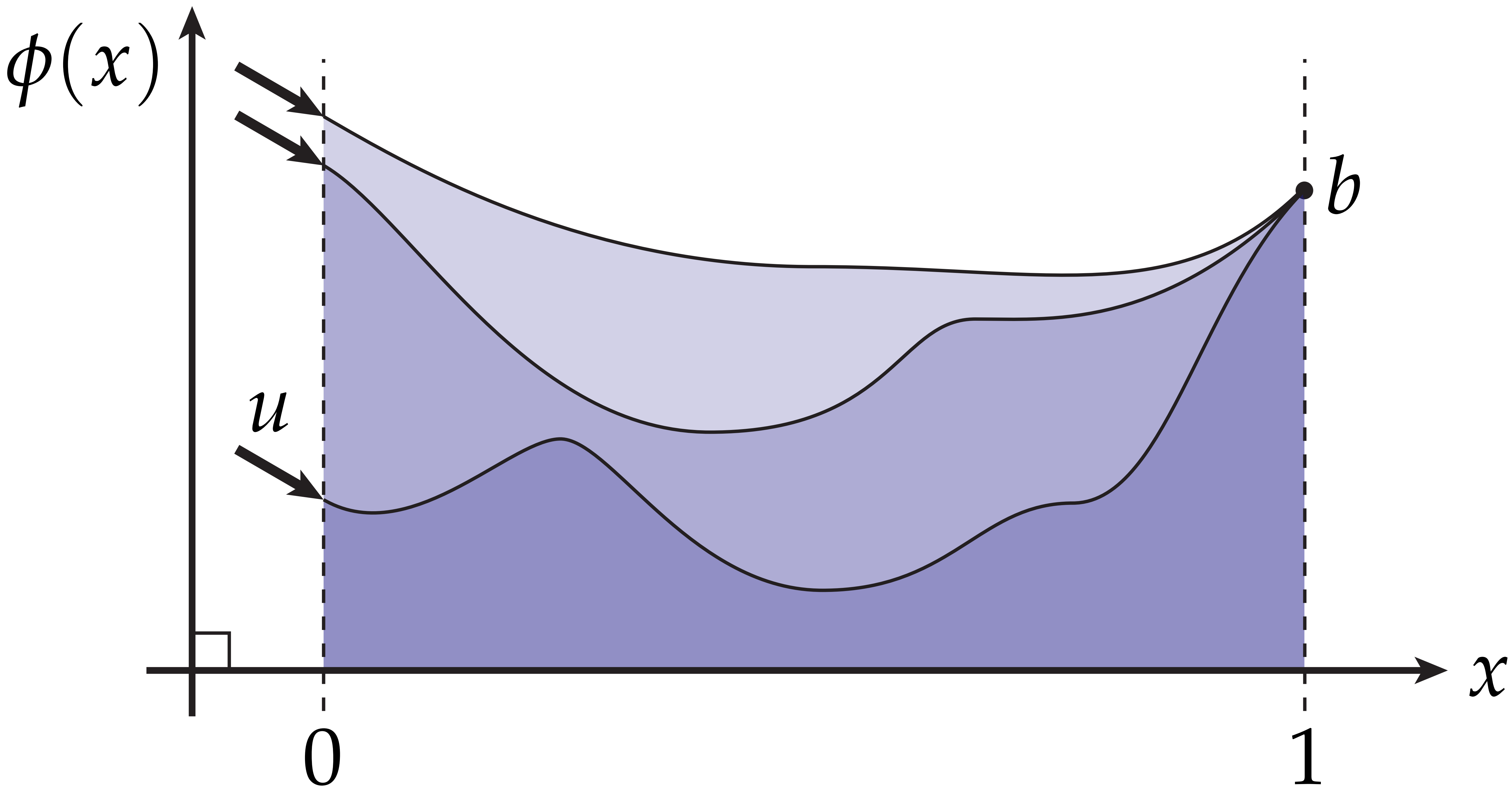
- **Neumann means "prescribe derivatives"**
- **E.g.,** $\phi'(0) = u$**,** $\phi'(1) = v$



- **Again, many possible functions!**

Both Neumann & Dirichlet

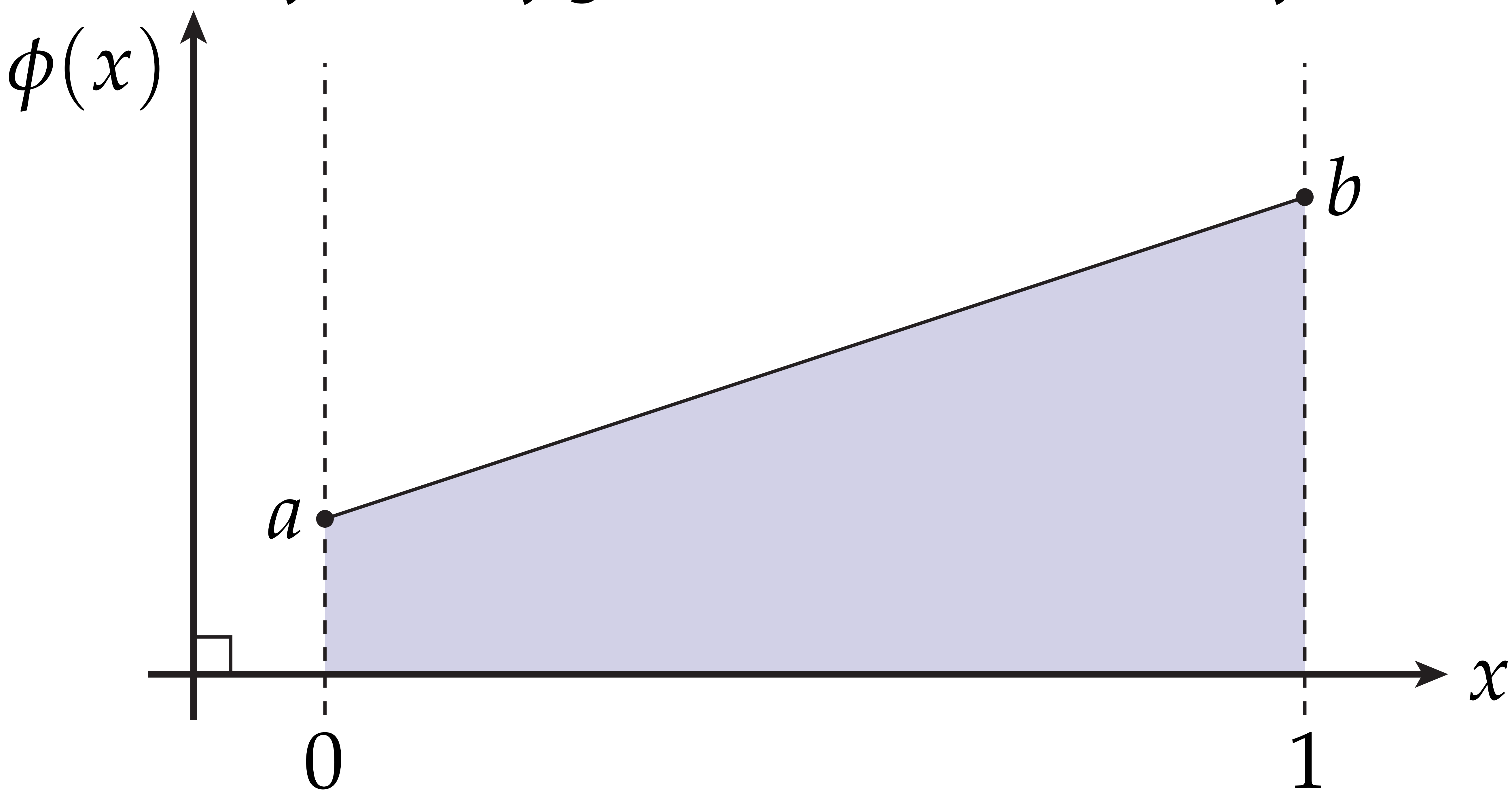
- **Or: prescribe some values, some derivatives**
- **E.g.,** $\phi'(0) = u$**,** $\phi(1) = b$



- **Q: What about** $\phi'(1) = v$**,** $\phi(1) = b$**? Does that work?**
- **Q: What about** $\phi'(0) + \phi(0) = p$**,** $\phi'(1) + \phi(1) = q$**? (Robin)**

1D Laplace w/ Dirichlet BCs

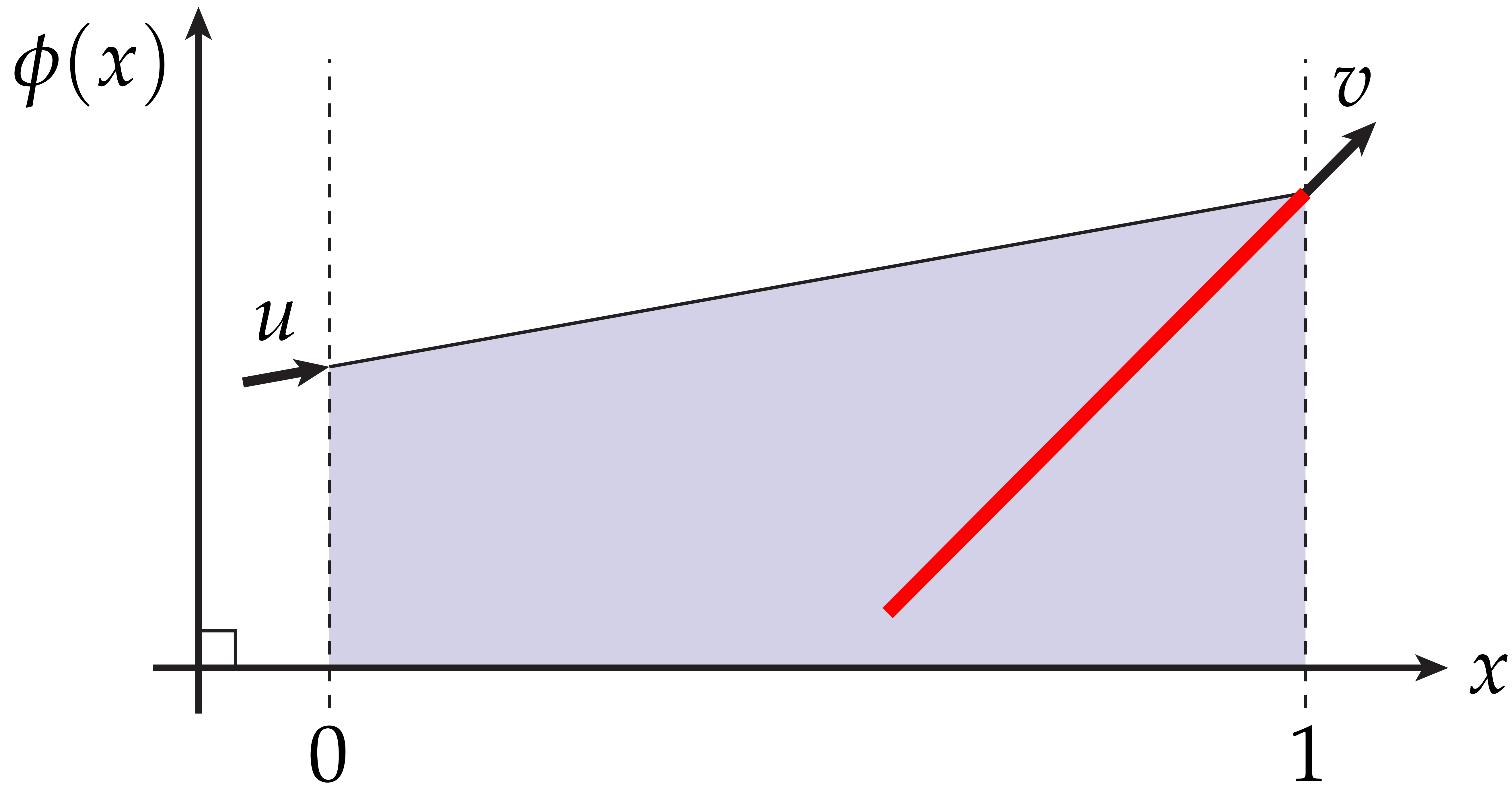
- **1D Laplace:** $\partial^2\phi/\partial x^2 = 0$
- **Solutions:** $\phi(x) = cx + d$
- **Q: Can we always satisfy given Dirichlet boundary conditions?**



- **Yes: a line can interpolate any two points.**

1D Laplace w/ Neumann BCs

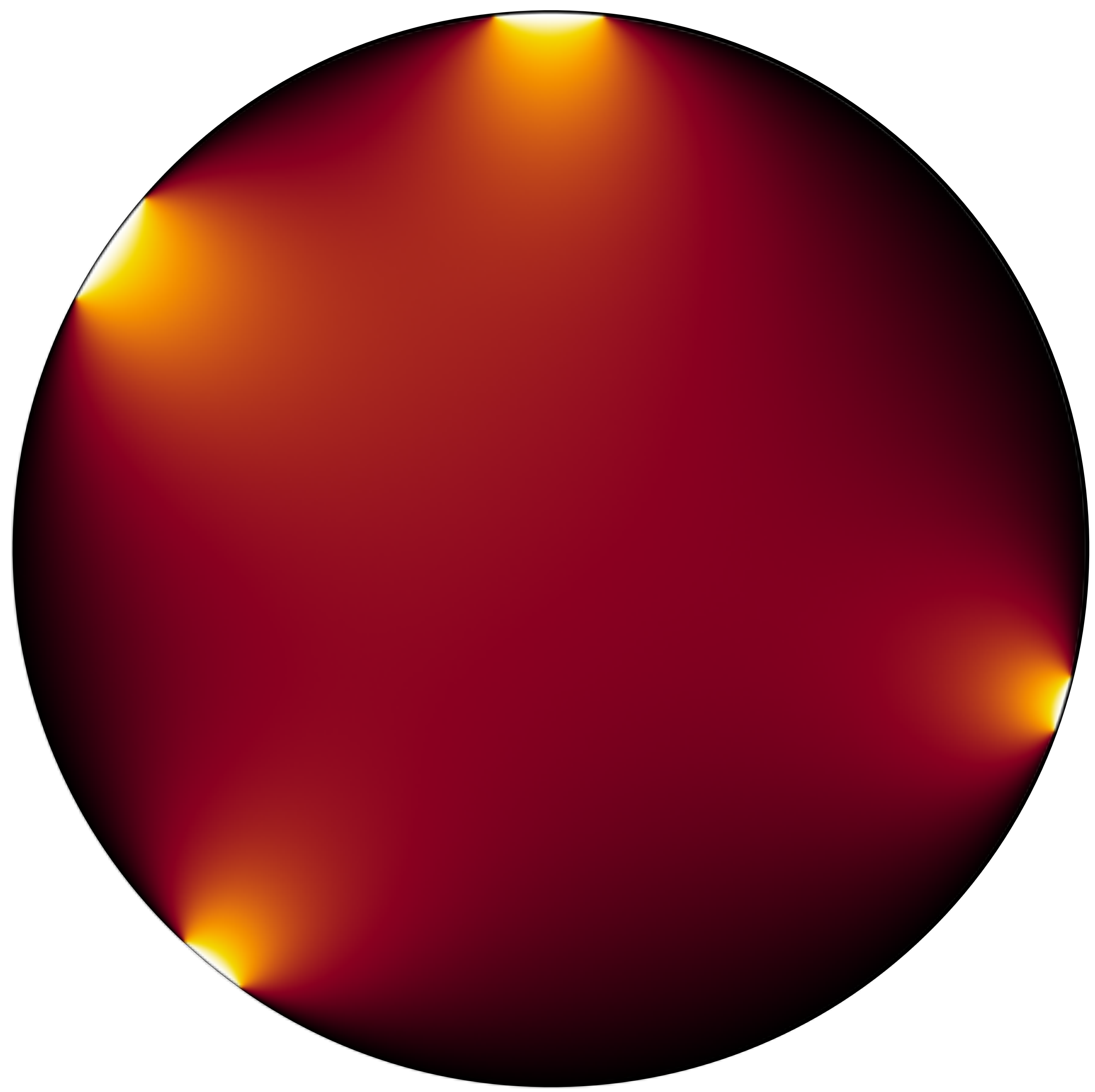
- **What about Neumann BCs?**
- **Q: Can we prescribe the derivative at both ends?**



- **No! A line has only one slope.**
- **In general, solution to a PDE may not exist for given BCs.**

2D Laplace w/ Dirichlet BCs

- **2D Laplace:** $\Delta\phi = 0$
- **Q: Can satisfy any Dirichlet BCs? (given data along boundary)**

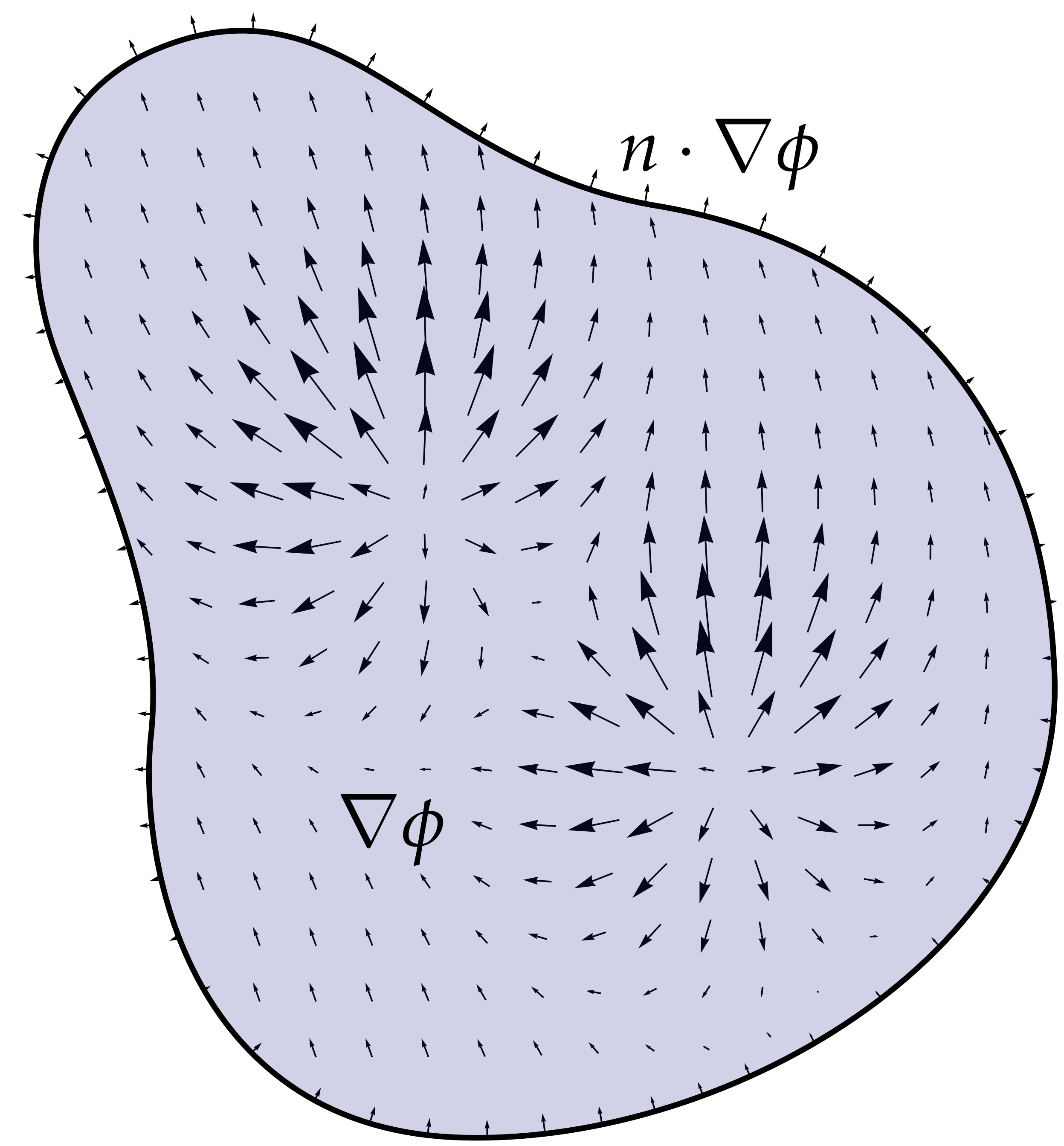


- **Yes: Laplace is long-time solution to heat flow**
- **Data is "heat" at boundary. Then just let it flow...**

2D Laplace w/ Neumann BCs

- **What about Neumann BCs for $\Delta\phi = 0$?**
- **Neumann BCs prescribe derivative in normal direction: $n \cdot \nabla\phi$**
- **Q: Can it always be done? (Wasn't possible in 1D...)**
- **In 2D, we have the divergence theorem:**

$$\int_{\partial\Omega} n \cdot \nabla\phi = \int_{\Omega} \nabla \cdot \nabla\phi = \int_{\Omega} \Delta\phi \stackrel{!}{=} 0$$



- **Should be called, "what goes in must come out theorem!"**
- **Can't have a solution unless the net flux through the boundary is zero.**
- **Numerical libraries will not always tell you if there's a problem!**
- **Trust, but verify (e.g., after solving $Ax = b$, compute $\|b - Ax\|$)**

Solving the Heat Equation

- **Back to our three model equations, want to solve heat eqn.**

$$\dot{u} = \Delta u$$

- **Just saw how to discretize Laplacian**
- **Also know how to do time (forward Euler, backward Euler, ...)**
- **E.g., forward Euler:**

$$u^{k+1} = u^k + \tau \Delta u^k$$

- **Q: On a grid, what's our overall update now at $u_{i,j}$?**

$$u_{i,j}^{k+1} = u^k + \tfrac{\tau}{h^2}(4u_{i,j}^k - u_{i+1,j}^k - u_{i-1,j}^k - u_{i,j+1}^k - u_{i,j-1}^k)$$

- **Not hard to implement! Loop over grid, add up some neighbors.**

Solving the Wave Equation

- **Finally, wave equation:**

$$\ddot{u} = \Delta u$$

- **Not much different; now have 2nd derivative in time**
- **By now we've learned two different techniques:**
  - **Convert to two 1st order (in time) equations:**

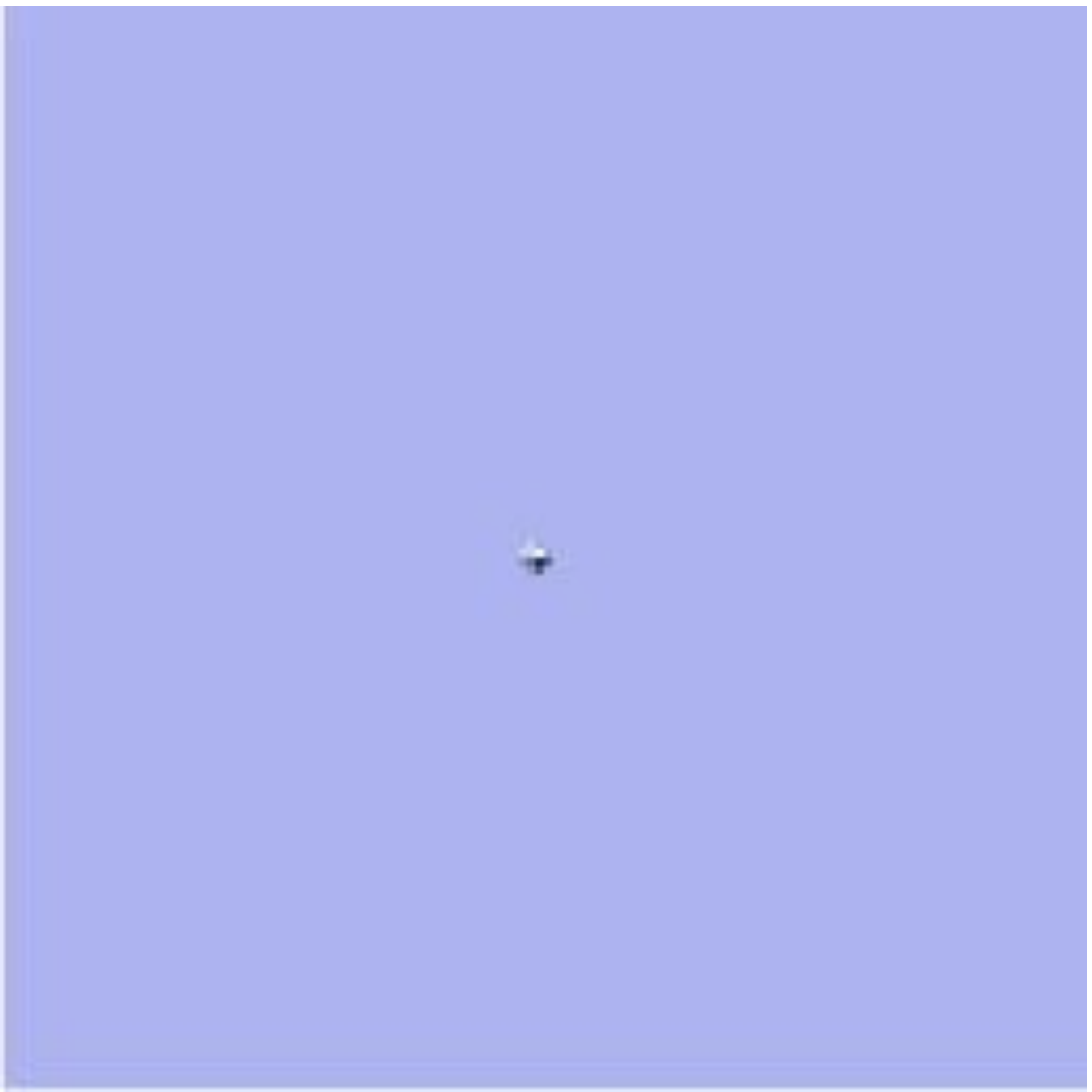
$$\dot{u} = v, \quad \dot{v} = \Delta u$$

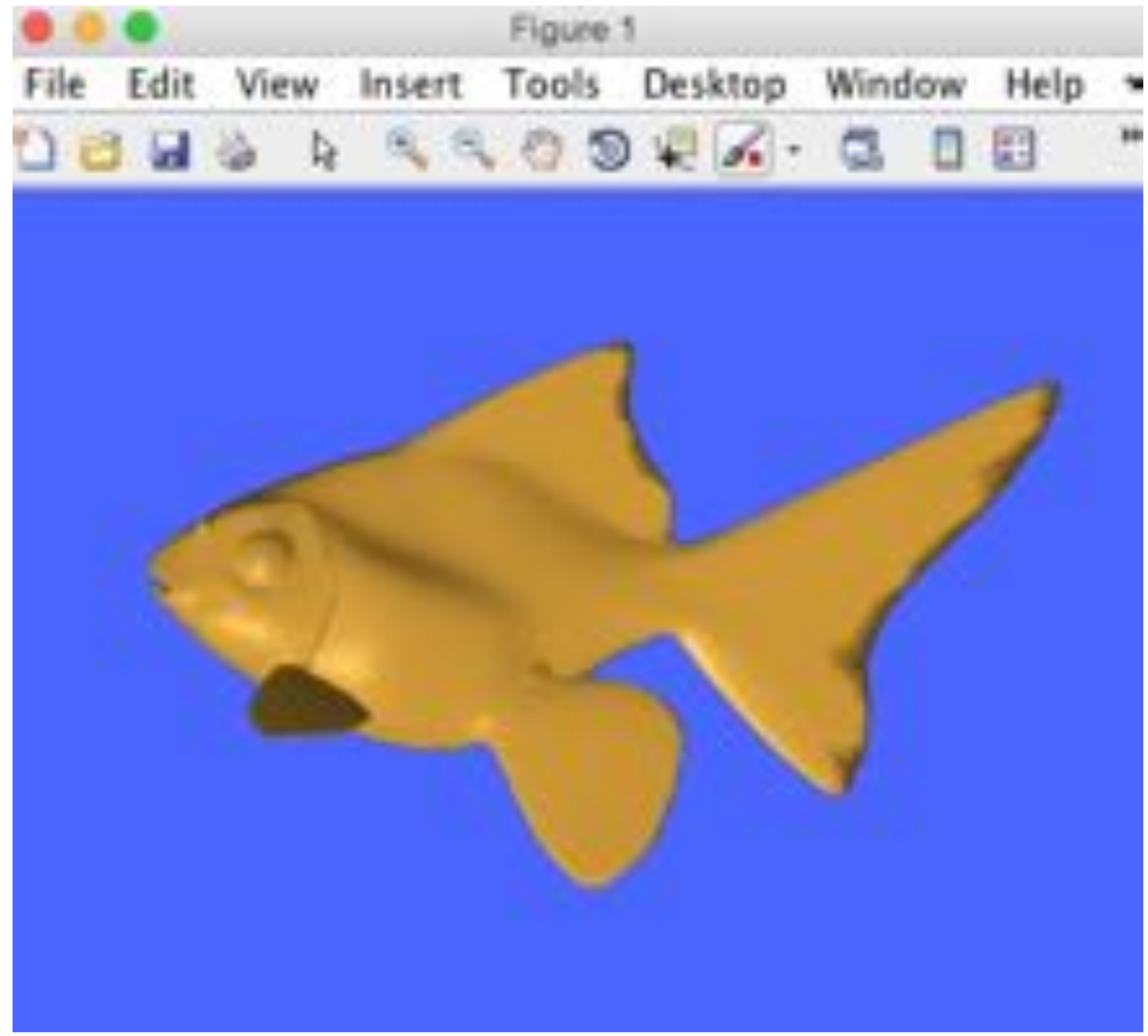
  - **Or, use centered difference (like Laplace) in time:**

$$\frac{u^{k+1} - 2u^k + u^{k-1}}{\tau^2} = \Delta u^k$$

- **Plus all our choices about how to discretize Laplacian.**
- **So many choices! And many, many (many) more we didn't discuss.**

Wave Equation on a Grid, Triangle Mesh





Fish credit: Alec Jacobson (http://www.alecjacobson.com/weblog/?p=4363)

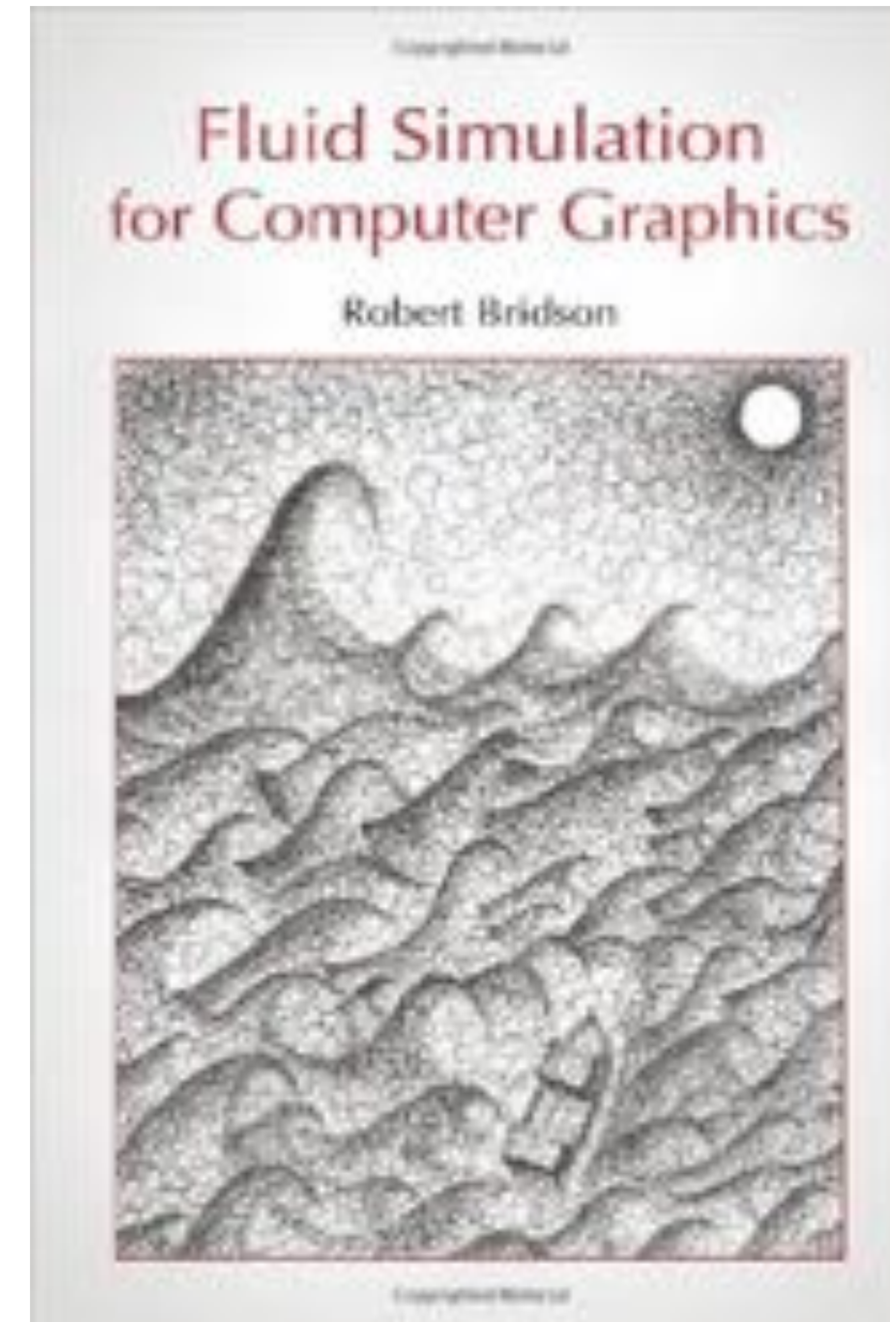
Fun with wave-like equations…

Technique: low-res thin shell simulation (via "position-based dynamics") + Loop subdivision

Wait, what about all that other cool stuff?
(Fluids, hair, cloth, …)

Want to Know More?

- There are some good books:



- And papers:

http://www.physicsbasedanimation.com/

- Also, what did the folks who wrote these books & papers read?