Lecture 22: Convergence of Iterative Methods

July 20, 2025

Lecture 22: Convergence of Iterative Methods - Outline

Convergence of Iterative Methods - Introduction

- In this final lecture of new material for which you will be responsible, we will examine the convergence of the conjugate gradient method in more detail.
- We then introduce the idea of **preconditioning** to accelerate convergence of the conjugate gradient method.
- The conjugate gradient method was first introduced in Lecture 09.

Convergence of Iterative Methods

Assumption: A is SPD.

We will need the following fact soon. For any k,

$$\begin{aligned} e^{(k)} &= x - x^{(k)}, \text{ so that} \\ e^{(0)} &= x - x^{(0)}, \text{ and} \\ Ae^{(0)} &= A\left(x - x^{(0)}\right) \\ &= Ax - Ax^{(0)} \\ &= b - Ax^{(0)} \\ &= r^{(0)}. \end{aligned}$$

Recall, at each step, CG finds the best solution $x^{(k)}$ in the span of search vectors so far, under the A-norm. That is, the CG method minimizes

$$\left\| e^{(k)} \right\|_A = \left\| x - x^{(k)} \right\|_A = \min_{x' \in \mathcal{K}_k(A)} \left\| x - x' \right\|_A,$$

where x is the true solution and $\mathcal{K}_k$ is a Krylov subspace (see Lecture 09).

Convergence of Iterative Methods

We can write

$$
\begin{aligned}
\left\| e^{(k)} \right\|_A &= \min \left\| x - \underbrace{\left(x^{(0)} + \sum_{i=0}^{k-1} \alpha_i p^{(i)} \right)}_{x' \in \mathcal{K}_k(A)} \right\|_A, \text{ for } \alpha_i \in \mathbb{R} \\
&= \min \left\| x - x^{(0)} - \sum_{i=0}^{k-1} \alpha_i p^{(i)} \right\|_A \\
&= \min \left\| e^{(0)} - \sum_{i=0}^{k-1} \alpha_i p^{(i)} \right\|_A \\
&= \min \left\| e^{(0)} + \sum_{i=0}^{k-1} \gamma_i A^{(i)} r^{(0)} \right\|_A, \text{ for } \gamma_i \in \mathbb{R},
\end{aligned}
$$

since $\text{span}\{p^{(0)}, p^{(1)}, \ldots, p^{(k-1)}\} = \text{span}\{r^{(0)}, Ar^{(0)}, A^2 r^{(0)}, \ldots, A^{k-1} r^{(0)}\}$, as in Lecture 09.

Convergence of Iterative Methods

Now define the following polynomial function

$$Q_{k-1}(x) = \gamma_0 + \gamma_1 x + \gamma_2 x^2 + \ldots + \gamma_{k-1} x^{k-1}.$$

Then taking the matrix A as the argument we have

$$\begin{aligned} Q_{k-1}(A) &= \gamma_0 + \gamma_1 A + \gamma_2 A^2 + \ldots + \gamma_{k-1} A^{k-1}, \\ &= \sum_{i=0}^{k-1} \gamma_i A^i. \end{aligned}$$

Now we can rewrite the error as

$$\begin{aligned} e^{(k)} &= e^{(0)} + \sum_{i=0}^{k-1} \gamma_i A^i r^{(0)} \\ &= e^{(0)} + Q_{k-1}(A) r^{(0)}, \\ &= e^{(0)} + Q_{k-1}(A) A e^{(0)}, \text{ since } r^{(0)} = A e^{(0)}, \text{ as above} \\ &= (\underbrace{I + Q_{k-1}(A) A}_{\text{Another polynomial}}) e^{(0)}. \end{aligned}$$

Convergence of Iterative Methods

Define $P_k(x) = 1 + Q_{k-1}(x)x$, then $\deg(P_k) \leq k$ and $P_k(0) = 1$.
So, we have $e^{(k)} = P_k(A)e^{(0)}$ and therefore

$$\begin{aligned} & \|e^{(k)}\|_A \\ = \quad & \min\left\{ \left\| P_k(A)e^{(0)} \right\|_A : P_k(x) = \text{poly. of deg. } \leq k \text{ with } P_k(0) = 1 \right\}. \end{aligned}$$

That is, if $\tilde{P}_k(x)$ is any polynomial of degree $\leq k$ with $\tilde{P}_k(0) = 1$ then $\left\|e^{(k)}\right\|_A \leq \left\|\tilde{P}_k(A)e^{(0)}\right\|_A$. So CG finds (implicitly) the optimal polynomial to minimize the error in the A-norm. By choosing a particular polynomial we can obtain a bound on the error, given in the next theorem (see Shewchuk's article for expanded derivation).

Theorem 1

$$\left\|e^{(k)}\right\|_A \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k \left\|e^{(0)}\right\|_A.$$

[Proof still needed]

Convergence of Iterative Methods

The actual CG convergence depends on all eigenvalues and is often better. The above is a worst case upper bound. Let us consider A with only 3 distinct eigenvalues

$$\lambda_1 < \lambda_2 < \lambda_3 \quad \text{, some eigenvalue may have mulitplicity } > 1.$$

We can show convergence in 3 iterations by choosing a "good" polynomial!

Convergence of Iterative Methods

- Write the initial error in terms of unit orthogonal, eigenvectors v_j, of A

$$e^{(0)} = \sum_{j=1}^{n} \xi_j v_j, \text{ for some coefficients } \xi_i.$$

- Also observe $P_k(A)v_j = P_k(\lambda_j)v_j$ (See Lecture Notes).
- Now, form a **Lagrange polynomial** $P_3(x)$ with degree ≤ 3 such that $P_3(0) = 1$ and $P_3(\lambda_j) = 0$ for $j = 1, 2, 3$.

Convergence of Iterative Methods

Note that

$$
\begin{aligned}
e^{(k)} &= P_k(A) \sum_{j=1}^{n} \xi_i v_j, \\
&= \sum_{j=1}^{n} \xi_j P_k(\lambda_j) v_j, \text{ since } P_k(A) v_j = P_k(\lambda_j) v_j \\
\Rightarrow Ae^{(k)} &= \sum_{j=1}^{n} \xi_j P_k(\lambda_j) \lambda_j v_j, \text{ since } v_j \text{ is an eigenvector of } A \\
\Rightarrow \left\| e^{(k)} \right\|_A^2 &= \sum_{j=1}^{n} \xi_j^2 \left[P_k(\lambda_j) \right]^2 \lambda_j, \text{ by the orthonormality of the } v_j s
\end{aligned}
$$

Convergence of Iterative Methods

Hence,

$$\left\| e^{(3)} \right\|_A^2 \underbrace{\leq}_{\text{earlier observation, for } P_3} \left\| P_3(A) e^{(0)} \right\|_A^2$$

$$\underbrace{=}_{\text{See Lecture Notes}} \sum_{j=1}^{n} \xi_j^2 \left[\underbrace{P_3(\lambda_j)}_{=0} \right]^2 \lambda_j$$

$$= 0,$$

which means the error will be zero after 3 iterations. Since CG picks the optimal polynomial, it converges at least as fast as this, i.e., in 3 steps regardless of $\kappa(A)$.

Preconditioning Idea

- We have seen that depending on eigenvalues of A, convergence of CG may still be rather slow.
- For the discretization of the Poisson equation, the asymptotic convergence of SOR (with optimal ω) matches CG. (Note that CG has the advantage of not having to find any optimal parameter.)
- How can we improve the speed of convergence for CG even further?
- We speed up convergence using the idea of **preconditioning**.
- We want to find a **modified** linear system, with **nicer properties**, but has the **same solution**.
- We want a better condition number $\kappa(A)$ (or more clustered eigenvalues), to achieve faster convergence, i.e., fewer CG iterations.

Preconditioning Idea

Consider the system

$$Ax = b \quad \text{versus} \quad (M^{-1}A)x = M^{-1}b.$$

The same x is a solution to both problems, but we would prefer to solve the "easier" one. In this situation the matrix M is called a **preconditioner**. Similar to splitting in stationary iterative methods, we desire

1. $M \approx A$, (**exercise**: why?)
2. M^{-1} to be easy to build, or rather, $My = c$ to be cheap to solve. (**exercise**: why?)

Preconditioning Idea - Symmetric Preconditioning

- To use CG, we need our modified system to also be SPD.
- Note that $M^{-1}A$ is not necessarily SPD, even if M and A are.
- If we let M be SPD, then a Cholesky factorization $M = LL^T$ exists.
- We instead form a new modified system as

$$\underbrace{L^{-1}AL^{-T}}_{\tilde{A}}\underbrace{L^T x}_{\tilde{x}} = \underbrace{L^{-1}b}_{\tilde{b}}.$$

Preconditioning Idea

Notice $\tilde{A}$ is SPD by construction. The preconditioner is effectively **split** into left and right parts.
Claim: $\tilde{A}$ is similar to $M^{-1}A$.
Proof: Observe that

$$\begin{aligned} M^{-1}A &= \left(LL^T\right)^{-1} A \\ &= L^{-T}L^{-1}A, \text{ so that} \\ L^T\left(M^{-1}A\right)L^{-T} &= L^T\left(L^{-T}L^{-1}A\right)L^{-T} \\ &= L^{-1}AL^{-T} \\ &= \tilde{A}, \end{aligned}$$

and hence $\tilde{A}$ is a similarity transform of $M^{-1}A$.
Moral: This **could** now be solved by CG and this system has the same convergence behavior, since $M^{-1}A$ and $L^{-1}AL^{-T}$ have the same eigenvalues!

Preconditioning Idea

This leads to a naïve approach for preconditioning CG. The naïve approach is:

- form the modified system, $L^{-1}AL^{-T}\tilde{x} = \tilde{b}$,
- apply basic CG,
- transform solution $\tilde{x}$ to recover x (solve $L^T x = \tilde{x}$).

The downside of this naïve approach is that it requires factoring M, which is potentially very costly. We must also run the full LL^T process on it, possibly leading to a large fill.

Common Preconditioners

- A better approach would be to not form $\tilde{A}$ explicitly.
- Instead we modify the CG algorithm itself (via change of variables) to include a single new "preconditioning step"

$$x^k = M^{-1} r^k.$$

- The theory only requires M to be SPD and we must be able to solve $Mz^k = r^k$ (hopefully cheaply).
- With this better approach, there is no need for factorization of $M = LL^T$.
- The preconditioned CG method is given in Algorithm 1.
- Note that we essentially add one extra line; if $M = I$, we recover basic CG.

Common Preconditioners

Algorithm 1 Preconditioned CG Algorithm

$x^0 = \text{initial guess}$

$r^0 = b - Ax^0$

for $k = 0, 1, 2, \ldots, n-1$

$z^k = M^{-1} r^k$ (or preferably solve $Mz^k = r^k$)

$$\beta^k = \begin{cases} 0 & \text{if } k = 0 \\ \frac{(z^k, r^k)}{(z^{k-1}, r^{k-1})} & \text{otherwise} \end{cases}$$

$$p^k = \begin{cases} z^k & \text{if } k = 0 \\ z^k + \beta^k p^{k-1} & \text{otherwise} \end{cases}$$

$\alpha^k = \frac{(z^k, r^k)}{(p^k, Ap^k)}$

$x^{k+1} = x^k + \alpha^k p^k$

$r^{k+1} = r^k - \alpha^k Ap^k$

end for

Common Preconditioners

Common preconditioners for M often are related to our stationary iterative methods:

- Jacobi preconditioning:

$$M_J = D \qquad \text{(easiest! but not great)},$$

- Symmetric Gauss-Seidel:

$$M_{SGS} = (D - L)D^{-1}(D - U),$$

- Symmetric SOR:

$$M_{SSOR} = (D - \omega L)D^{-1}(D - \omega U).$$

Common Preconditioners - SGS Implementation

One can express

$$M_{SGS} = (D - L)D^{-1}(D - U) = L_M U_M,$$

as an LU factorization where

$$\begin{aligned} L_M &= (D - L)D^{-1}, \\ U_M &= (D - U). \end{aligned}$$

Since L_M is unit lower triangular and U_M is upper triangular, SGS preconditioning $z^k = M^{-1}r^k$ just requires two triangular solves:

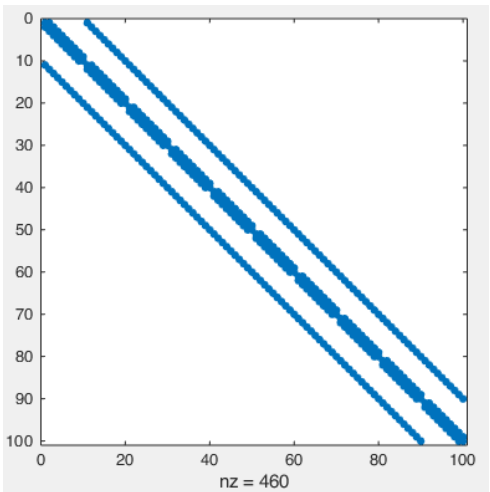
$$\begin{aligned} (I - LD^{-1})y &= r^k, \\ (D - U)z^k &= y. \end{aligned}$$

Common Preconditioners - "Incomplete" Cholesky preconditioning

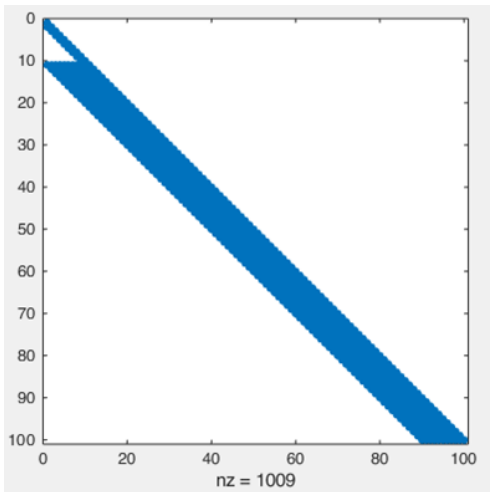
- The $L_M U_M$ factorization above gives us the hint for using other factorizations.
- With incomplete Cholesky (IC) preconditioning we find a partial Cholesky factorization where $LL^T \approx A$ (only **approximately**).
- We construct L via a Cholesky-like process, but skip (some or all) steps that would introduce new non-zero entries.
- The IC preconditioner is not guaranteed to exist except in special cases (e.g., Laplacian, other M-matrices, etc.).

Common Preconditioners - "Incomplete" Cholesky preconditioning

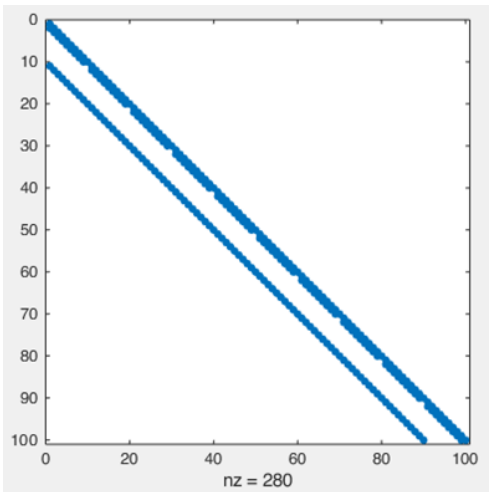
The figure below shows an example of the sparsity pattern of the Cholesky and IC factorizations of the discrete Laplacian.



Laplacian A



Cholesky Factor



Incomplete Cholesky Factor

The sparsity pattern of L in the IC factorization stays close to A's compared to the full Cholesky factorization. Therefore, memory/speed cost remains low, but eigenvalues improve enough to accelerate CG convergence significantly.

Common Preconditioners - "Incomplete" Cholesky preconditioning

For example, with the discrete Laplacian for $m = 14$

$\kappa(A) \approx 90.5, \lambda_{max} \approx 1780, \lambda_{min} \approx 19.7, 23$ CG iterations for $tol = 10^{-7}$.

However, setting $L = \texttt{ichol}(A)$ and $\tilde{A} = L^{-1}AL^{-T}$ we have

$\kappa(\tilde{A}) \approx 8.9, \lambda_{max} \approx 1.2, \lambda_{min} \approx 0.135, 14$ PCG iterations for $tol = 10^{-7}$.

Notice, we now have a better condition number, smaller eigenvalues, and fewer iterations for convergence.

Common Preconditioners - "Incomplete" Cholesky preconditioning

- MATLAB's CG routine is `pcg` for preconditioned conjugate gradient.
- It accepts preconditioner(s) as extra arguments. Incomplete Cholesky preconditioning is supported via `ichol`.
- The demo code `PCGDemo.m` compares CG, PCG, and optimal SOR for solving the Laplace equation.
- It can be seen from running this code that PCG converges much faster than the other two methods.

Extensions

A big limitation to the CG method is that it only applies to SPD matrices. However, many matrices encountered "in the wild" are not of this form. We briefly discuss how non-SPD matrices are handled in this section.

Option #1: One could solve the linear system as a least-squares problem. The solution to $\min_x \|Ax - b\|_2^2$ for square A satisfies $Ax = b$. So we just need to solve the normal equations $A^T Ax = A^T b$ with CG ("CGNR"). For this approach:

- Simple to code and $A^T A$ is SPD!
- The condition is much worse ($\approx$ squared).

Option #2: Extensions of CG ideas ("Krylov solvers") exist for general systems:

- Symmetric **indefinite** systems: MINRES, SYMMLQ, ...,
- General **non-symmetric**: GMRES, BiCGSTAB, ...

Similar to CG, these aim to satisfy certain optimality properties. For example, MINRES seeks to minimize the norm of the residual.

Extensions

For preconditioning there are also many others such as:

- (sparse) approximate inverse preconditioners,
- multilevel/multigrid preconditioners,
- parallel preconditioners,
- domain decomposition and block preconditioners, etc.

Preconditioners can also be applied to Krylov methods for indefinite and non-symmetric linear systems (MINRES, GMRES, etc.). Finding effective preconditioners for Krylov methods can depend heavily on the specific application problem/domain and its matrix structure. For more, see Preconditioning Techniques for Large Linear Systems: A Survey [Benzi 2002].

(Last) Graphics Application

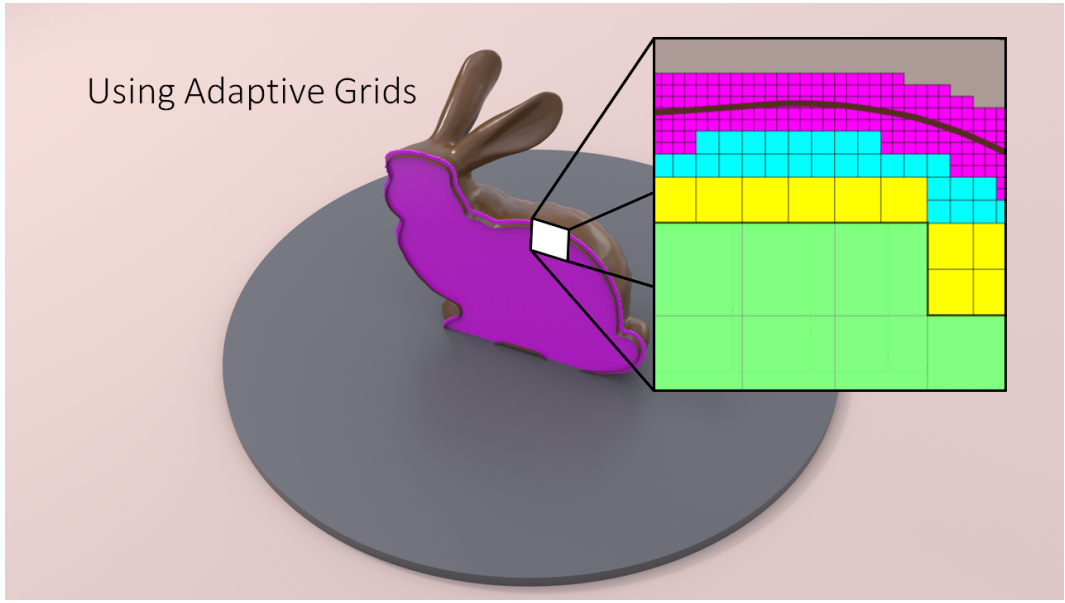
Dr. Christopher Batty here at UWaterloo (and some of his students) enjoy animating viscous liquids. However, the linear systems are very large and denser than Poisson/Laplacian. In one example, of melting the Stanford Bunny, the cost of solving the linear system for the viscosity costs **way more** ($\approx 95\%$ of total) than any other step of fluid animation!
Reducing the cost has been tackled in two ways:

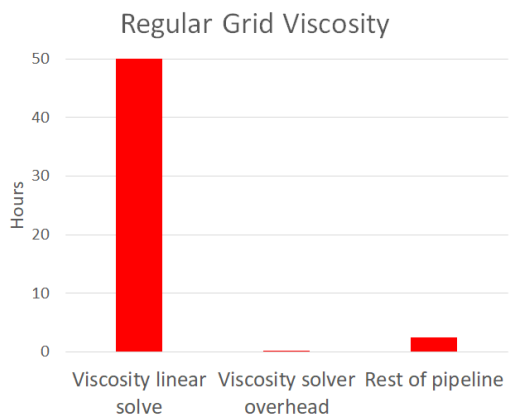
- Adaptive grid structures,
- Specialized multigrid preconditioners.

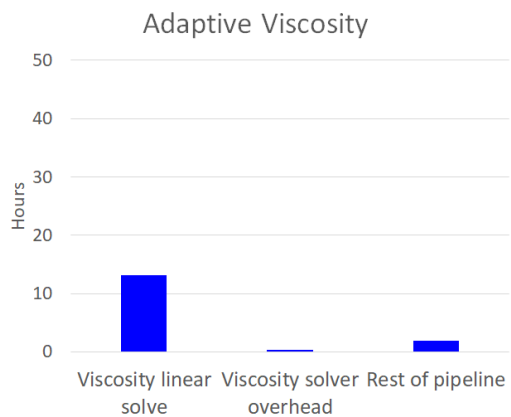
(Last) Graphics Application

- Adaptive grid structures reduce the cost by adding fine grids only where fine detail is necessary.
- In graphics, the interesting visual details are usual near the boundary.
- Therefore, fine grids are used near the boundary and larger grids are used far away from the boundary.
- The size of the overall linear system is therefore smaller compared to using the fine grid throughout the whole domain.

(Last) Graphics Application

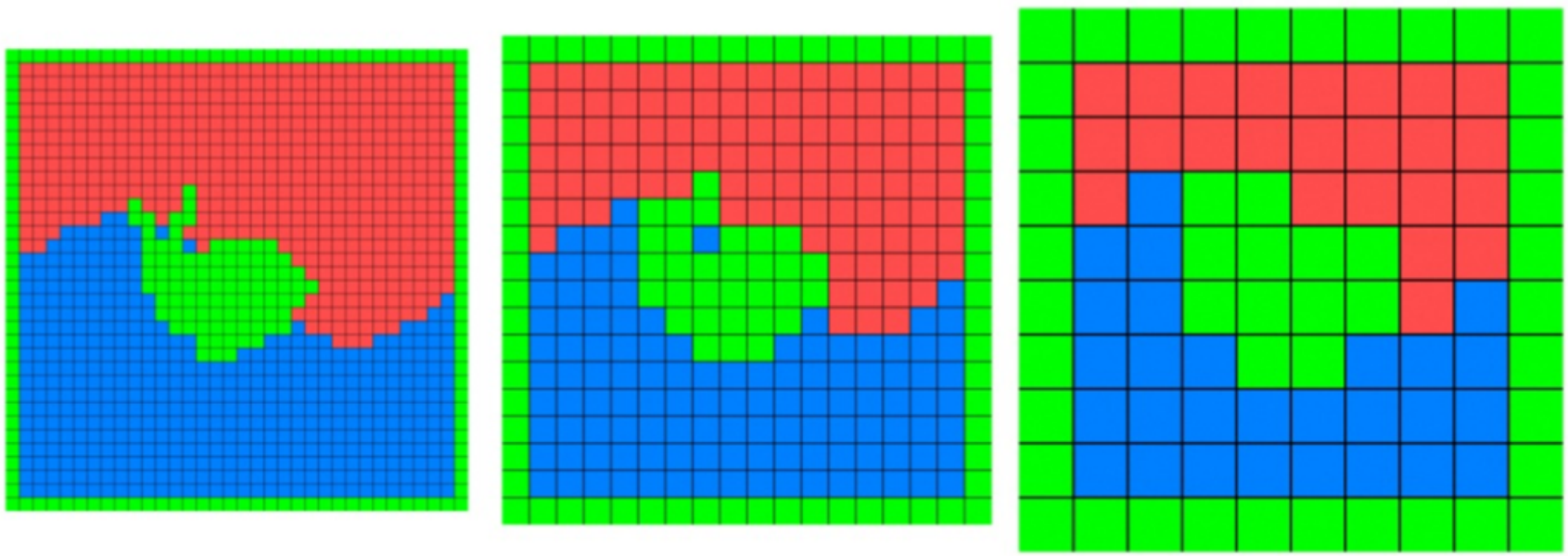






(Last) Graphics Application

The idea of using a **multigrid preconditioner** is as follows. One creates a multi-level approximation of the physical domain, perform "smoothing" at each level (using local Cholesky factorizations), and use the whole process as part of a CG preconditioner.



(Last) Graphics Application

Solver	128^3 grid 3,035,346 DOFs	256^3 grid 24,303,388 DOFs	512^3 grid 194,503,048 DOFs
Multigrid (Naïve baseline)	20 iterations 377.3 s 0.256 GB 2 levels	34 iterations 975.3 s 2.18 GB 3 levels	39 iterations 2920.4 s 15.94 GB 4 levels
Multigrid (Improved baseline)	21 iterations 90.1 s 0.256 GB 2 levels	36 iterations 333.8 s 2.18 GB 3 levels	41 iterations 1512.4 s 15.94 GB 4 levels
Multigrid (Ours)	3 iterations 27.3 s 0.256 GB 2 levels	3 iterations 119.6 s 2.18 GB 3 levels	4 iterations 834.7 s 15.94 GB 4 levels
Eigen	377 iterations 35.6 s 1.536 GB	707 iterations 619.4 s 11.78 GB	out of memory
ICPCG	39 iterations 6.7 s 1.024 GB	53 iterations 95.3 s 8.32 GB	out of memory
PARDISO	1626.9 s 24.96 GB	N/A	N/A

(Last) Graphics Application

The moral of the story is that familiarity with numerical linear algebra can enable huge speedups by:

(a) Using existing algorithms more wisely,

(b) Developing specialized algorithms for your problem/matrix.