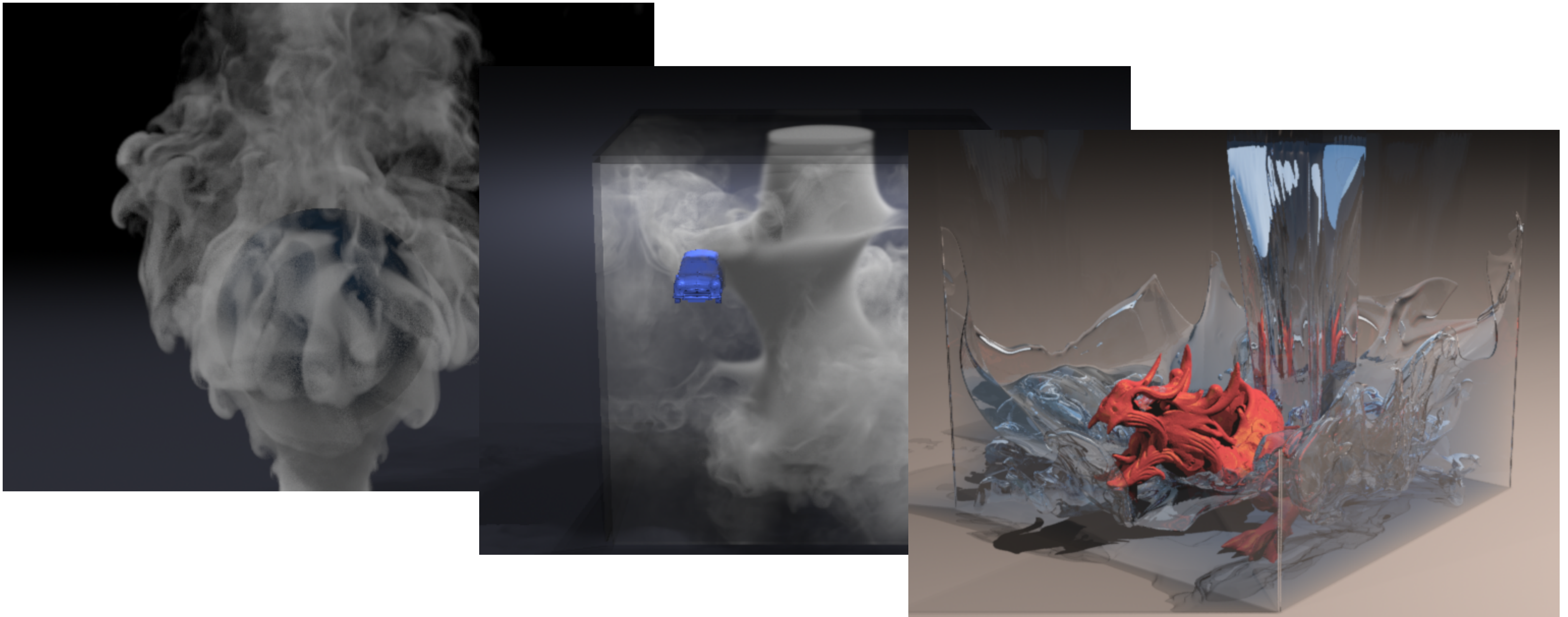


Stencil computations in the context of solving sparse linear systems
(motivated by computational physics and graphics)

Introduction

Example : The 3D Poisson equation

$$
\begin{pmatrix}
-6 & 1 & & & 1 & & & & 1 & & & \\
1 & -6 & 1 & & & 1 & & & & \ddots & & \\
 & 1 & -6 & 1 & & & \ddots & & & & \ddots & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & & 1 \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & 1 & \\
 & & & \ddots & & & \ddots & \ddots & \ddots & & & 1 \\
1 & & & & \ddots & & & \ddots & \ddots & \ddots & & \\
 & \ddots & & & & \ddots & & & 1 & -6 & 1 & \\
 & & \ddots & & & & 1 & & & 1 & -6 & 1 \\
 & & & 1 & & & & 1 & & & 1 & -6
\end{pmatrix}
\mathbf{x} = \mathbf{b}
$$

What about x & b?
How are they stored?

Introduction

Example : The 3D Poisson equation

$$\mathbf{x} = \begin{pmatrix} \text{u[0][0][0]} \\ \text{u[0][0][1]} \\ \vdots \\ \text{u[0][0][511]} \\ \text{u[0][1][0]} \\ \text{u[0][1][1]} \\ \vdots \\ \text{u[0][1][511]} \\ \vdots \\ \text{u[511][511][511]} \end{pmatrix}$$

$$\begin{pmatrix} \ddots & & \\ & \ddots & \\ & & 1 \\ \ddots & & \\ & 1 & \\ & & 1 \\ \ddots & & \\ -6 & 1 & \\ 1 & -6 & 1 \\ & 1 & -6 \end{pmatrix} \mathbf{x} = \mathbf{b}$$

What about x & b?
How are they stored?

Introduction

Example : The 3D Poisson equation

$$\mathbf{x} = \begin{pmatrix} \text{u} \\ \text{u} \\ \vdots \\ \text{u} \\ \text{u} \\ \text{u} \\ \vdots \\ \text{u} \\ \vdots \\ \text{u} \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} \text{Lu[0][0][0]} \\ \text{Lu[0][0][1]} \\ \vdots \\ \text{Lu[0][0][511]} \\ \text{Lu[0][1][0]} \\ \text{Lu[0][1][1]} \\ \vdots \\ \text{Lu[0][1][511]} \\ \vdots \\ \text{Lu[511][511][511]} \end{pmatrix}$$

$\mathbf{x} = \mathbf{b}$

What about x & b?
How are they stored?

Introduction

Example : The 3D Poisson equation

$$
\begin{pmatrix}
-6 & 1 & & & 1 & & & & 1 & & & \\
1 & -6 & 1 & & & 1 & & & & \ddots & & \\
 & 1 & -6 & 1 & & & \ddots & & & & \ddots & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & & 1 \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & 1 & \\
 & & & \ddots & & & \ddots & \ddots & \ddots & & & 1 \\
1 & & & & \ddots & & & \ddots & \ddots & \ddots & & \\
 & \ddots & & & & \ddots & & & 1 & -6 & 1 & \\
 & & \ddots & & & & 1 & & & 1 & -6 & 1 \\
 & & & 1 & & & & 1 & & & 1 & -6
\end{pmatrix}
\mathbf{x} = \mathbf{b}
$$

*Computing b = L*x is equivalent to executing*
ComputeLaplacian(u,Lu)

Applications

Governing equation and boundary conditions

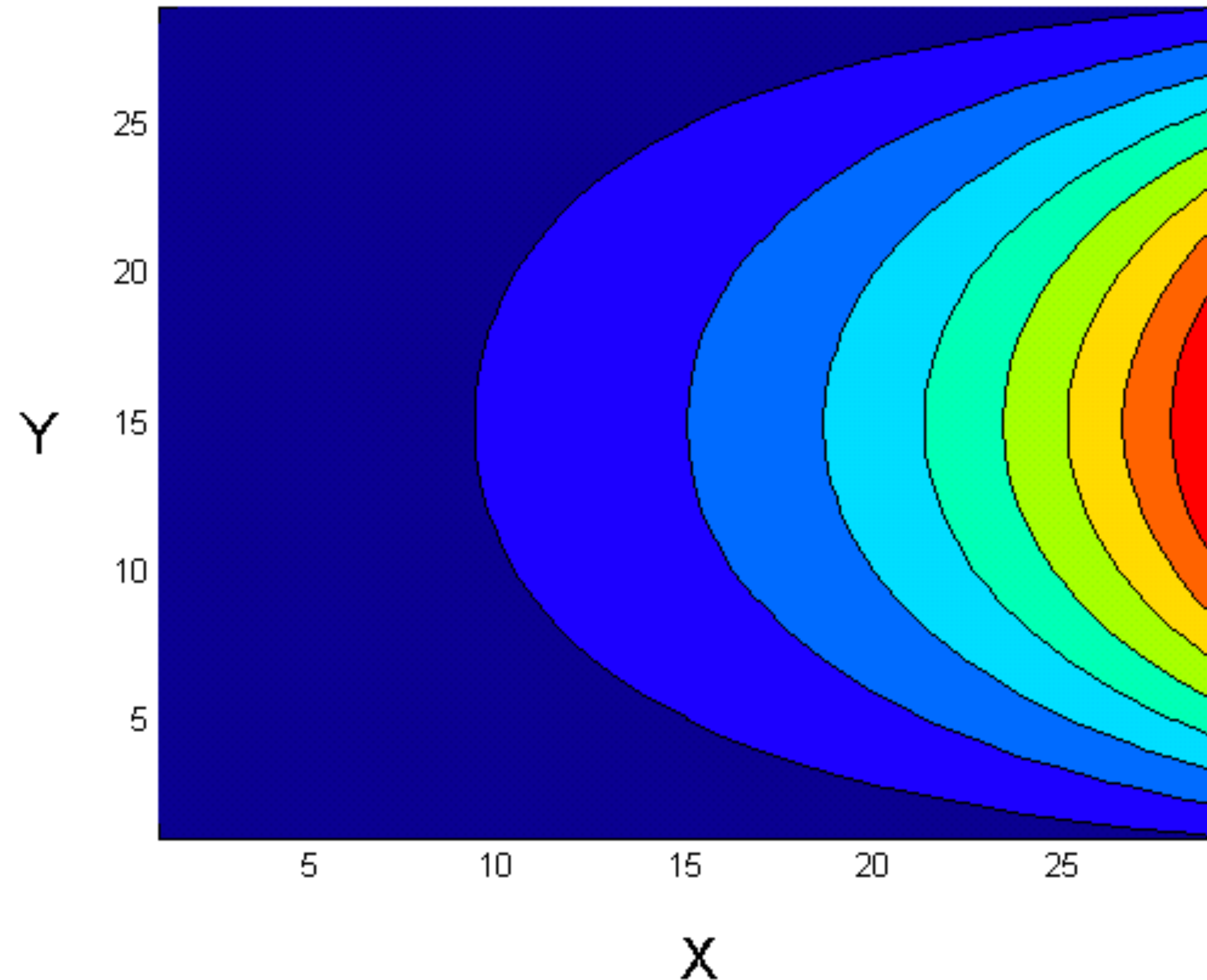
$$\nabla^2 u = u_{xx} + u_{yy} = 0$$

$$u(0,y) = 0 \quad 0 < y < 1$$

$$u(x,0) = u(x,1) = 0 \quad 0 < x < 1$$

$$u(1,y) = 100\sin(\pi y) \quad 0 < y < 1$$

Temperature profile in a rectangular plate



Finite difference scheme

$$A^{-1} * X = U$$

$$\mathbf{X} = \begin{pmatrix} \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} N-2 \\ -100\sin(\pi/N) \\ \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} N-2 \\ -100\sin(2\pi/N) \\ \vdots \\ \left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} N-2 \\ -100\sin(\frac{(N-1)}{N}\pi) \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} B & I & & & \\ I & B & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & B & I \\ & & & & I & B \end{pmatrix}, \quad \mathbf{I} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} -4 & 1 & & & \\ 1 & -4 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & -4 & 1 \\ & & & 1 & -4 \end{pmatrix}$$

Development Plan

Design

- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

- Implement a prototype
- Organize code into reusable kernels

Accelerate

- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels

Development Plan

Design

- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

- Implement a prototype
- Organize code into reusable kernels

Accelerate

- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels

Clarifying objectives

What kind of accuracy do we need?

- Solve $\boldsymbol{Ax=b}$ down to machine precision?
- Ensure that $\boldsymbol{x}$ is correct to k significant digits?
- Ensure that $\boldsymbol{x}$ (initial guess) is improved by k significant digits?

Clarifying objectives

What kind of accuracy do we need?

- Solve ***Ax=b*** down to machine precision?
- Ensure that ***x*** is correct to *k* significant digits?
- Ensure that ***x*** (initial guess) is improved by *k* significant digits?

What is the *real* underlying problem we care about?

- The system ***Ax=b*** is rarely the ultimate objective
- Typically, it's means to an end
  - Solve the system *to create a simulation*
  - Solve the system *to generate a solution to a physical law*

Clarifying objectives

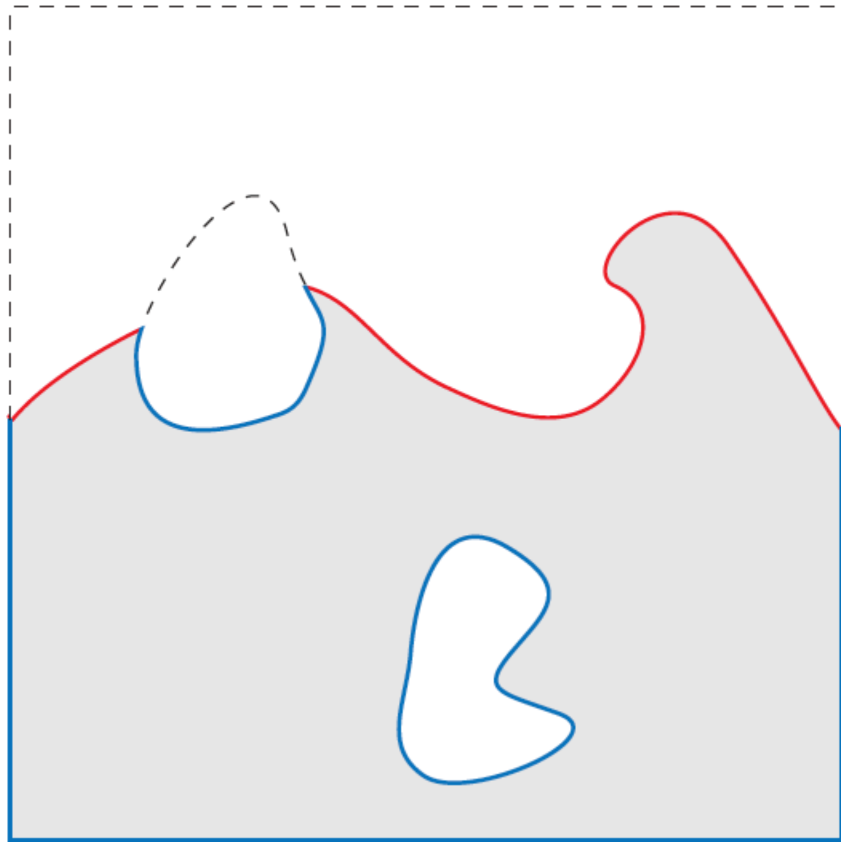
What kind of accuracy do we need?

- Solve ***Ax=b*** down to machine precision?
- Ensure that ***x*** is correct to *k* significant digits?
- Ensure that ***x*** (initial guess) is improved by *k* significant digits?

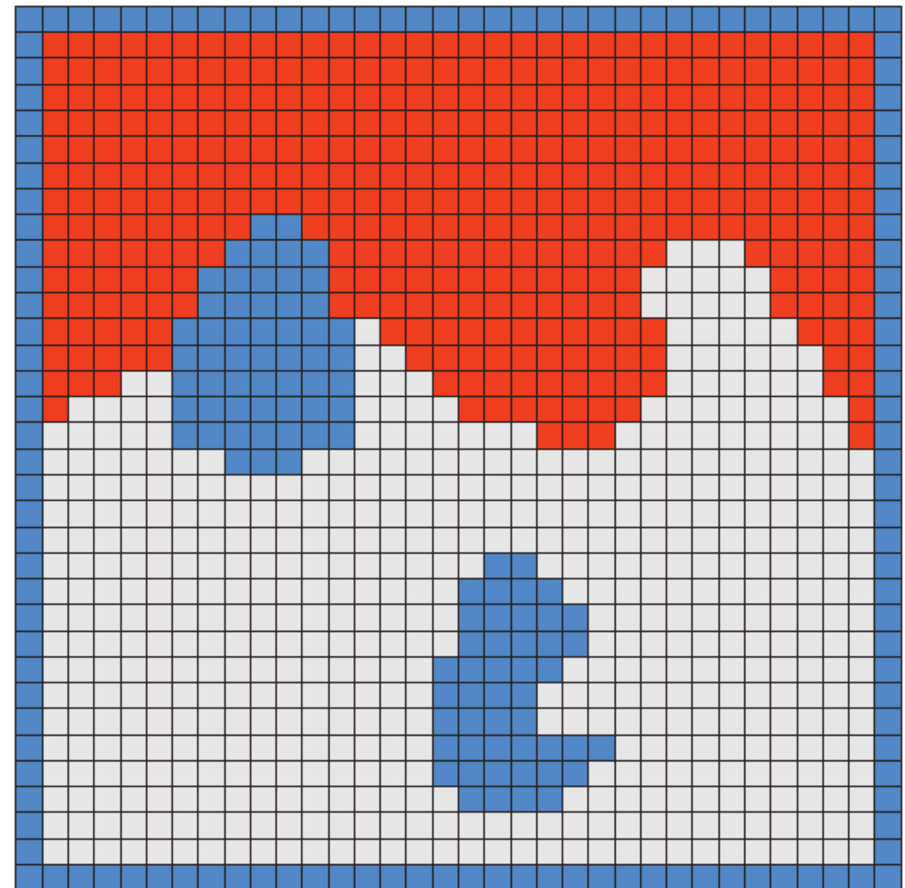
What is the *real* underlying problem we care about?

- The system ***Ax=b*** is rarely the ultimate objective
- Typically, it's means to an end
  - Solve the system *to create a simulation*
  - Solve the system *to generate a solution to a physical law*
- We have some flexibility to make ***Ax=b*** "better" for parallel algorithmic solution

Clarifying objectives

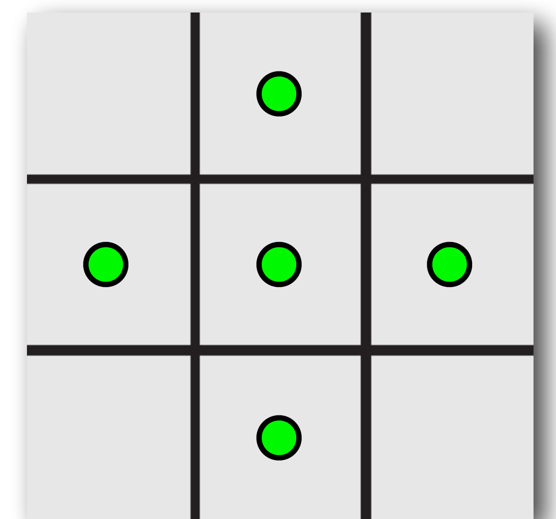


$$\Delta \mathbf{x} = \mathbf{b}$$



$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

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



Development Plan

Design

- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

- Implement a prototype
- Organize code into reusable kernels

Accelerate

- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels

Performance bounds and ***"textbook efficiency"***

Example : Solving the quadratic equation

$$ax^2 + bx + c = 0$$

What is the *minimum* amount of time needed to solve this?

Data access cost bound

*"We cannot solve this faster than the time needed to read **a,b,c** and write **x**"*

"We cannot solve this faster than the time needed evaluate the polynomial, for given values of a,b,c and x"
(i.e. 2 ADDs, 2 MULTs plus data access)

Solution verification bound

$$ax^2 + bx + c = (ax + b)x + c$$

Equivalent operation bound

"We cannot solve this faster than the time it takes to compute a square root"

Performance bounds and ***"textbook efficiency"***

What about linear systems of equations?

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

"Textbook Efficiency"
(for elliptic systems)

*It is **theoretically possible** to compute the solution to a linear system (with certain properties) with a cost comparable to **10x the cost of verifying** that a given value **x** is an actual solution*

... or ...

*It is **theoretically possible** to compute the solution to a linear system (with certain properties) with a cost comparable to **10x the cost of computing*** the expression ***r=b-Ax*** and verifying that ***r=0***
(i.e. slightly over 10x of the cost of a matrix-vector multiplication)

Development Plan

Design

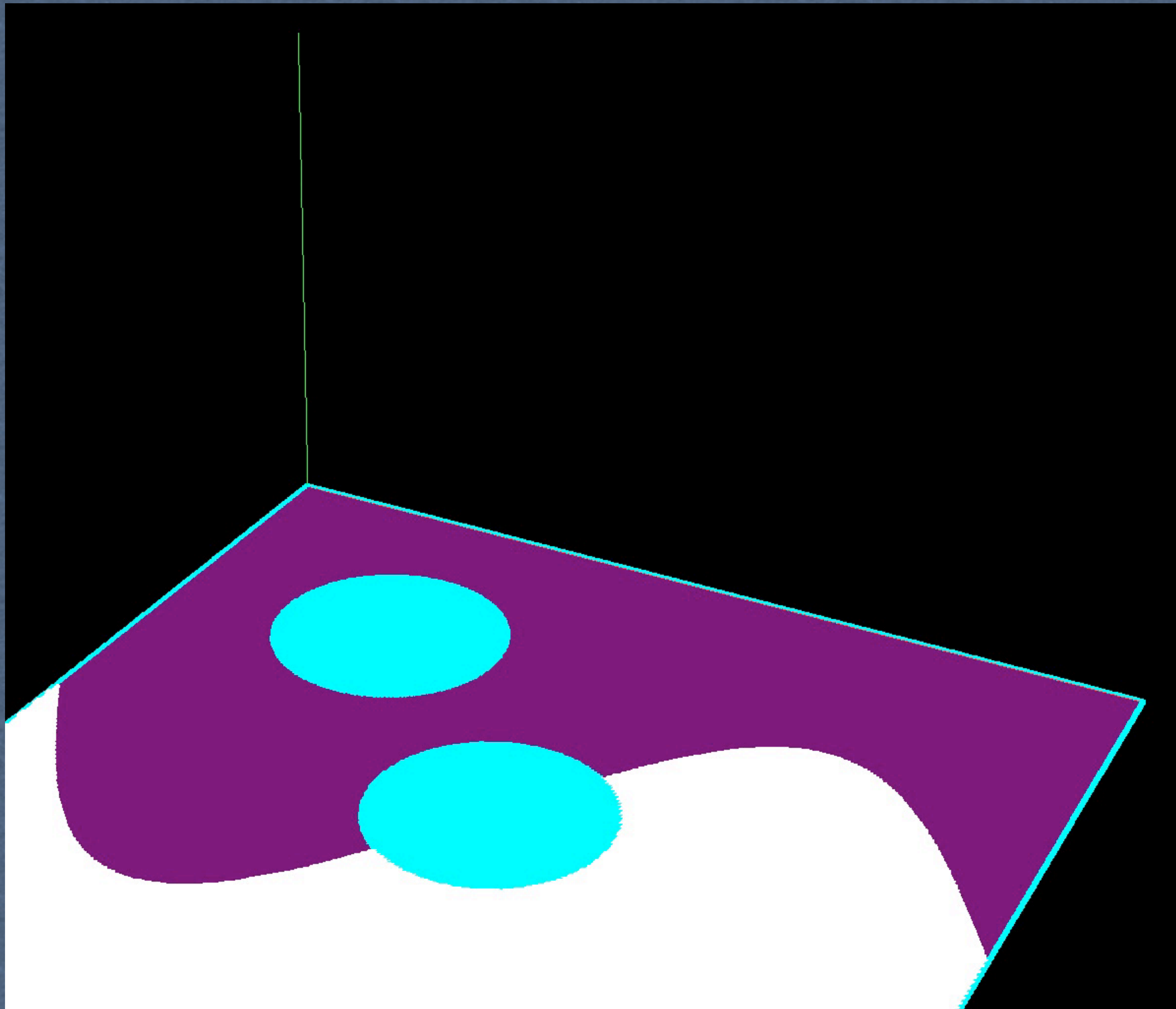
- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

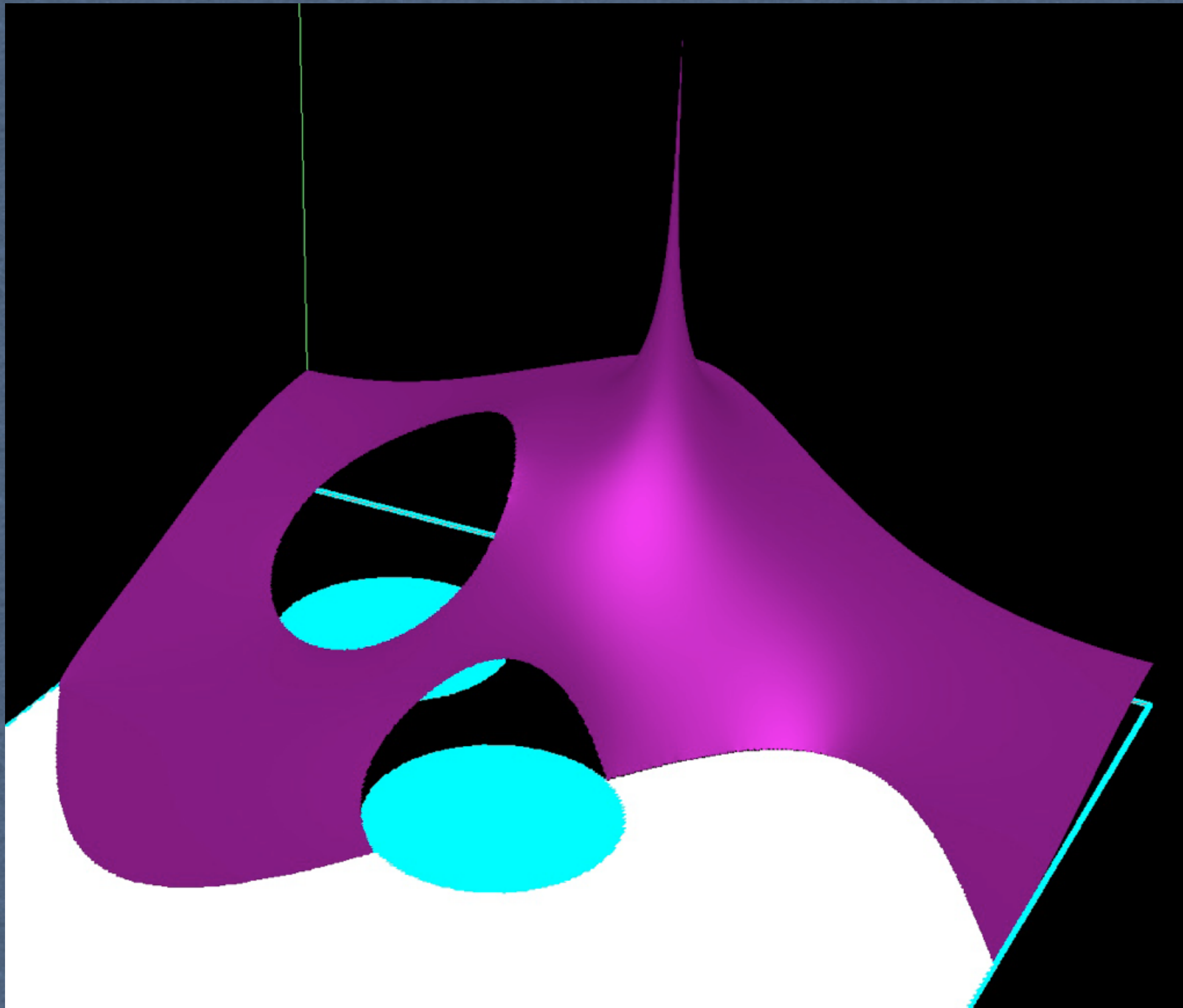
- Implement a prototype
- Organize code into reusable kernels

Accelerate

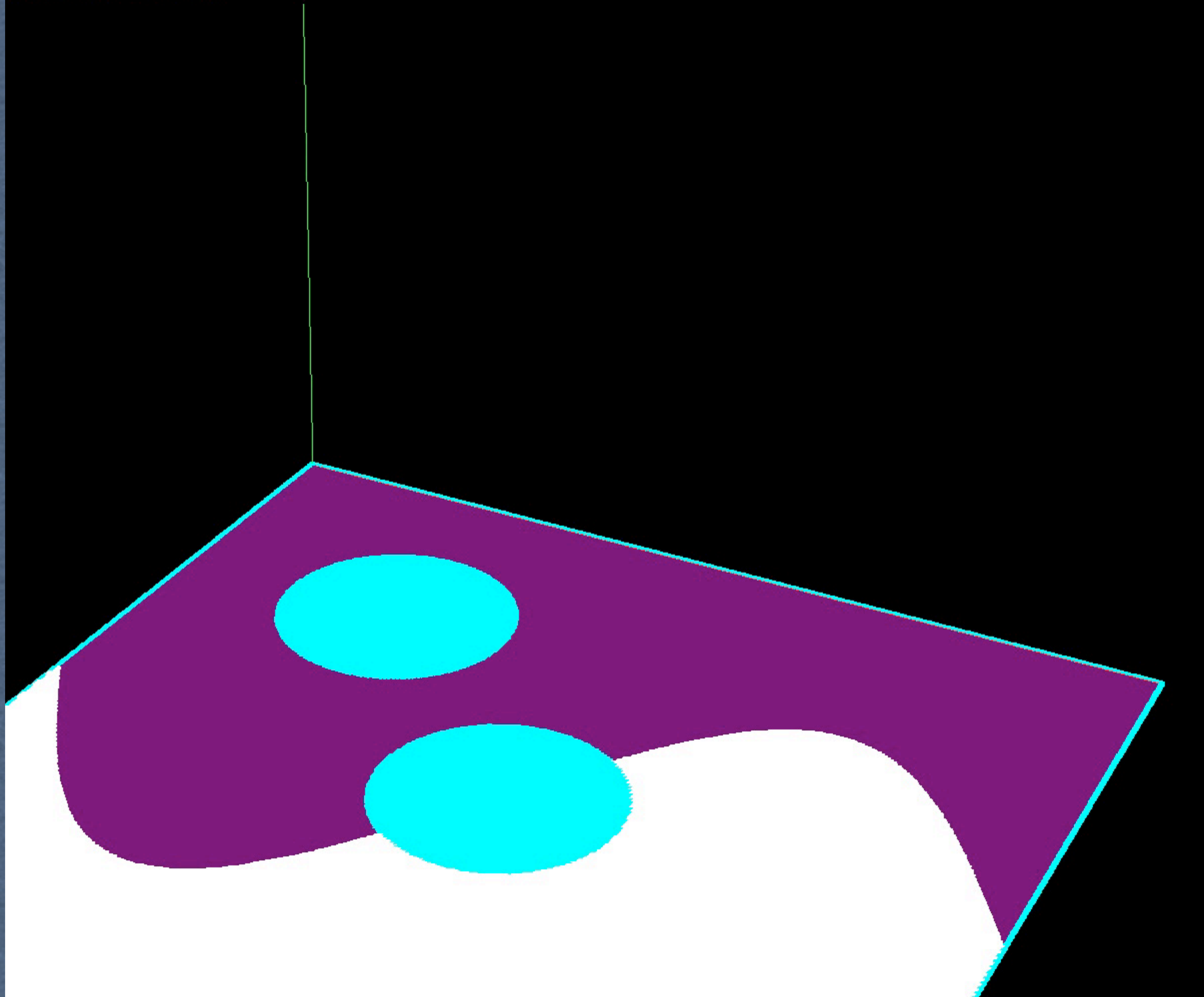
- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels



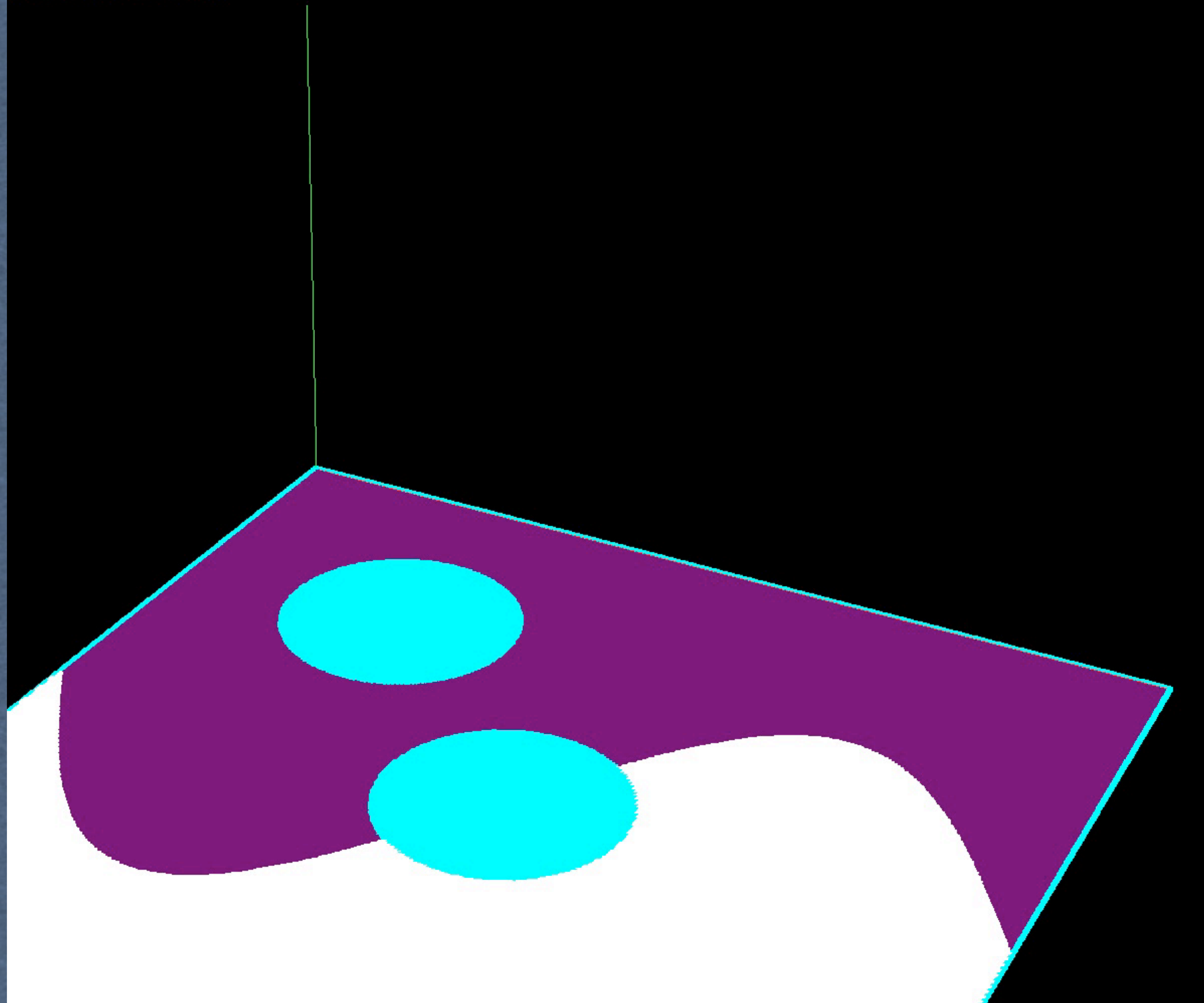
Sample 2D domain (512x512 resolution)



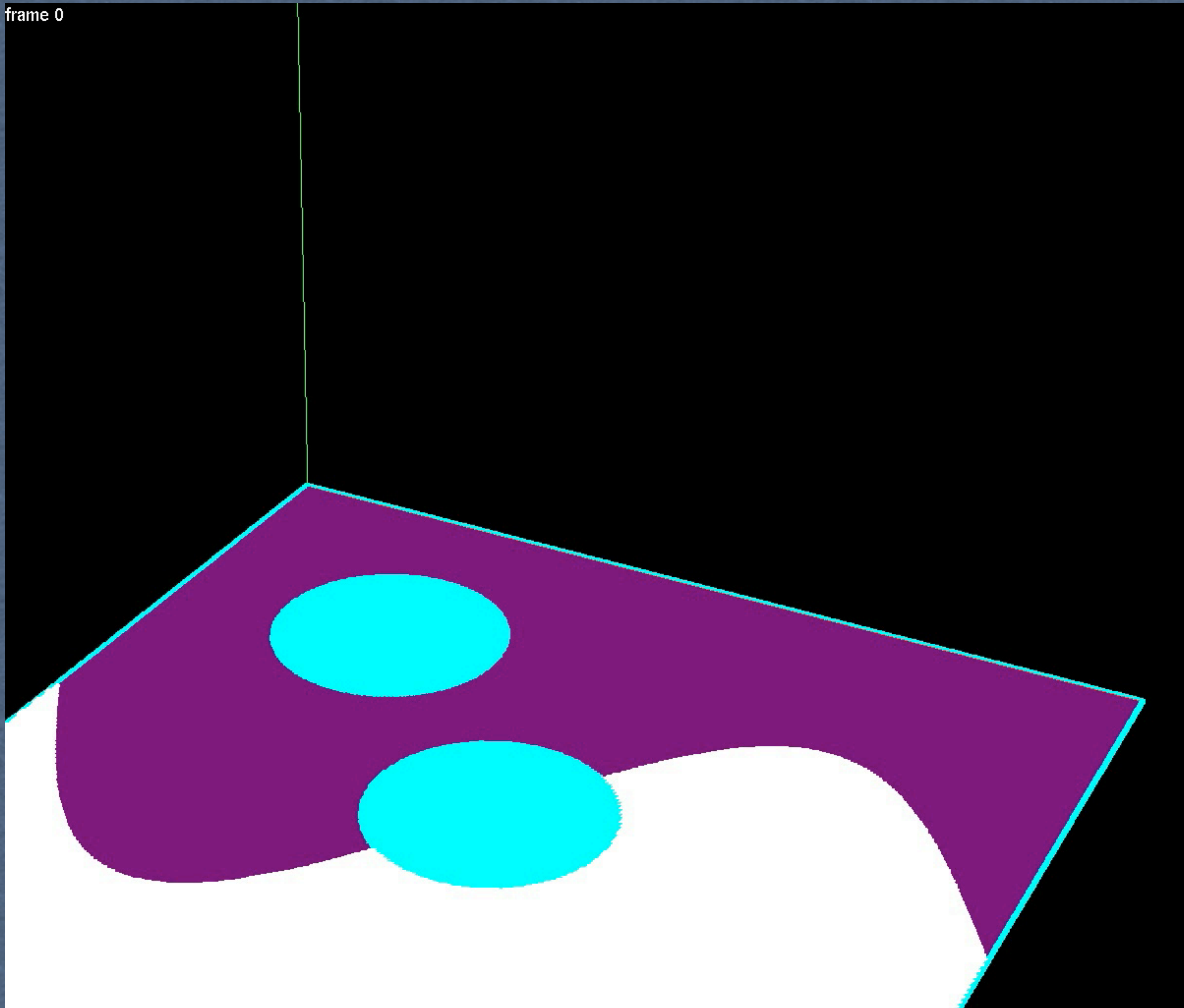
Exact solution



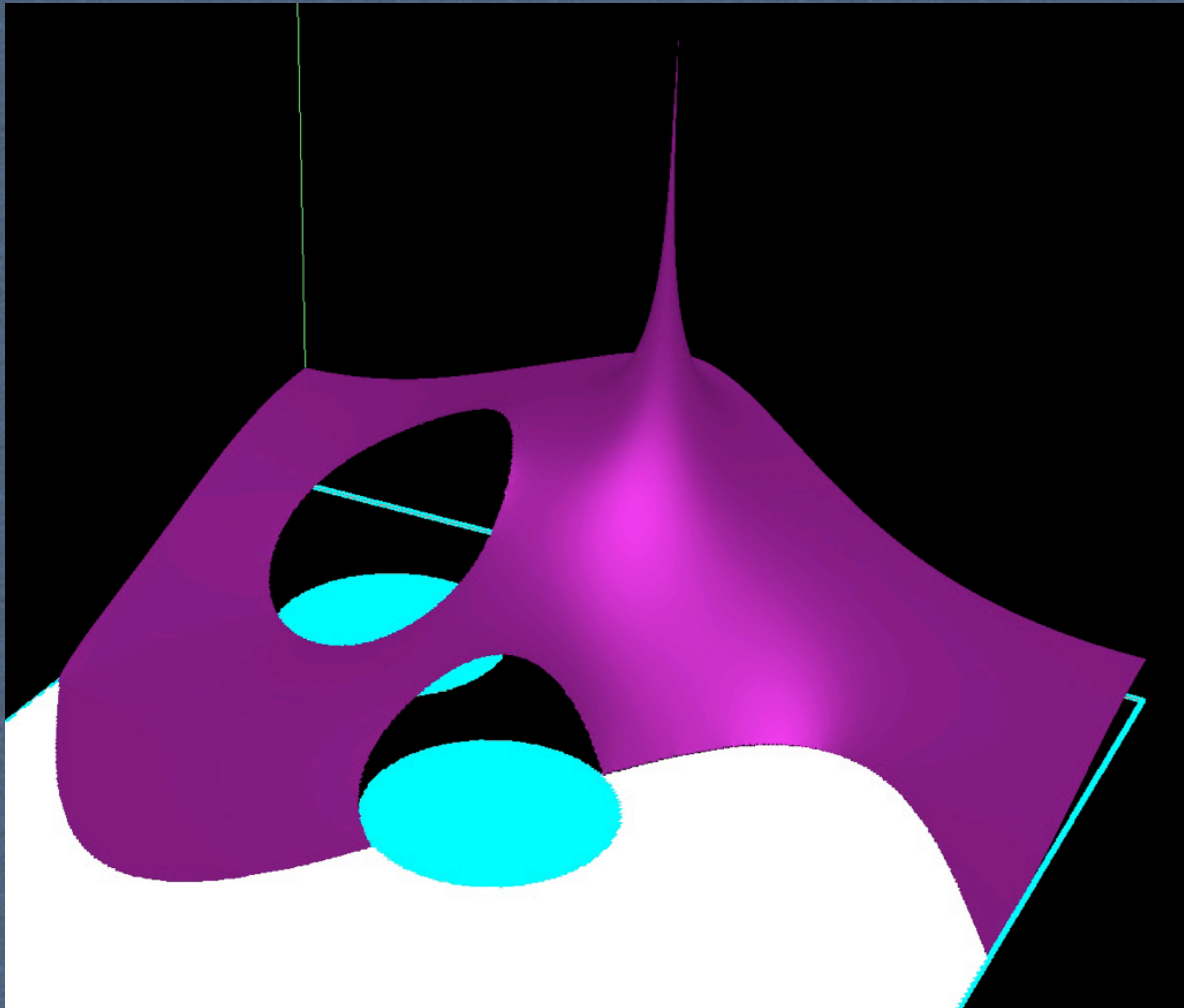
Conjugate Gradients (w/o preconditioning)



Conjugate Gradients (with a stock preconditioner)



Conjugate Gradients (with parallel multigrid preconditioner)



Exact solution

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Performance

- Converges in O(Nd) with stock preconditioners
- Converges in O(N) with multigrid preconditioners

Prerequisites

- Requires a symmetric system matrix
- Matrix needs to be positive definite
- (Other variants exist, too)

Benefits

- Low storage overhead
- Simple component kernels

1: **procedure** MGPCG($\mathbf{r},\mathbf{x}$)
2: $\quad \mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x},\ \mu \leftarrow \bar{\mathbf{r}},\ \nu \leftarrow \|\mathbf{r}-\mu\|_\infty$
3: $\quad$ **if** $(\nu < \nu_{\max})$ **then return**
4: $\quad \mathbf{r} \leftarrow \mathbf{r} - \mu,\ \mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)},\ \rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: $\quad$ **for** $k = 0$ **to** k_{max} **do**
6: $\quad\quad \mathbf{z} \leftarrow \mathcal{L}\mathbf{p},\ \sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\quad\quad \alpha \leftarrow \rho/\sigma$
8: $\quad\quad \mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z},\ \mu \leftarrow \bar{\mathbf{r}},\ \nu \leftarrow \|\mathbf{r}-\mu\|_\infty$
9: $\quad\quad$ **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\quad\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: $\quad\quad\quad$ **return**
12: $\quad\quad$ **end if**
13: $\quad\quad \mathbf{r} \leftarrow \mathbf{r} - \mu,\ \mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)},\ \rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\quad\quad \beta \leftarrow \rho^{\text{new}}/\rho$
15: $\quad\quad \rho \leftarrow \rho^{\text{new}}$
16: $\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p},\ \mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: $\quad$ **end for**
18: **end procedure**

Development Plan

Design

- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

- Implement a prototype
- Organize code into reusable kernels

Accelerate

- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

1: **procedure** MGPCG$(\mathbf{r}, \mathbf{x})$
2: $\quad \mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x},\ \mu \leftarrow \bar{\mathbf{r}},\ \nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: $\quad$ **if** $(\nu < \nu_{\max})$ **then return**
4: $\quad \mathbf{r} \leftarrow \mathbf{r} - \mu,\ \mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)},\ \rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: $\quad$ **for** $k = 0$ **to** k_{max} **do**
6: $\quad\quad \mathbf{z} \leftarrow \mathcal{L}\mathbf{p},\ \sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\quad\quad \alpha \leftarrow \rho/\sigma$
8: $\quad\quad \mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z},\ \mu \leftarrow \bar{\mathbf{r}},\ \nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: $\quad\quad$ **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\quad\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: $\quad\quad\quad$ **return**
12: $\quad\quad$ **end if**
13: $\quad\quad \mathbf{r} \leftarrow \mathbf{r} - \mu,\ \mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)},\ \rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\quad\quad \beta \leftarrow \rho^{\text{new}}/\rho$
15: $\quad\quad \rho \leftarrow \rho^{\text{new}}$
16: $\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p},\ \mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: $\quad$ **end for**
18: **end procedure**

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\beta \leftarrow \rho^{\text{new}}/\rho$
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Implemented as
ComputeLaplacian(p, z)

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

```
1:  procedure MGPCG(r, x)
2:      r ← r − 𝓛x, μ ← r̄, ν ← ||r − μ||_∞
3:      if (ν < ν_max) then return
4:      r ← r − μ, p ← 𝓜^{-1} r^{(†)}, ρ ← p^T r
5:      for k = 0 to k_max do
6:          z ← 𝓛p, σ ← p^T z
7:          α ← ρ/σ
8:          r ← r − αz, μ ← r̄, ν ← ||r − μ||_∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ, z ← 𝓜^{-1} r^{(†)}, ρ^new ← z^T r
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp, p ← z + βp
17:     end for
18: end procedure
```

Pointwise Ops (PointwiseOps.h)

LaplaceSolver/LaplaceSolver_0_0

```
#pragma once

#include "Parameters.h"

// Copy array x into y
void Copy(const float (&x)[XDIM][YDIM][ZDIM], float (&y)[XDIM][YDIM][ZDIM]);

// Scale array x by given number, add y, and write result into z
void Saxpy(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM][ZDIM],
    float (&z)[XDIM][YDIM][ZDIM], const float scale);
```

Pointwise Ops (PointwiseOps.cpp)

LaplaceSolver/LaplaceSolver_0_0

```
#include "PointwiseOps.h"

void Copy(const float (&x)[XDIM][YDIM][ZDIM], float (&y)[XDIM][YDIM][ZDIM])
{
#pragma omp parallel for
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        y[i][j][k] = x[i][j][k];
}

void Saxpy(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM][ZDIM],
    float (&z)[XDIM][YDIM][ZDIM],
    const float scale)
{
    // Should we use OpenMP parallel for here?
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        z[i][j][k] = x[i][j][k] * scale + y[i][j][k];
}
```

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\quad \mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: $\quad$ **if** $(\nu < \nu_{\max})$ **then return**
4: $\quad \mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: $\quad$ **for** $k = 0$ **to** k_{max} **do**
6: $\quad\quad \mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\quad\quad \alpha \leftarrow \rho/\sigma$
8: $\quad\quad \mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: $\quad\quad$ **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\quad\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: $\quad\quad\quad$ **return**
12: $\quad\quad$ **end if**
13: $\quad\quad \mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\quad\quad \beta \leftarrow \rho^{\text{new}}/\rho$
15: $\quad\quad \rho \leftarrow \rho^{\text{new}}$
16: $\quad\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: $\quad$ **end for**
18: **end procedure**

Implemented as
Saxpy(p, x, x, alpha)

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\beta \leftarrow \rho^{\text{new}}/\rho$
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Implemented as
Saxpy(p, z, p, beta)

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: … $\mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: … ρ^{new}/ρ
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Implemented as
Saxpy(z, r, r, -alpha)

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

```
1:  procedure MGPCG(r, x)
2:      r ← r − ℒx, μ ← r̄, ν ← ||r − μ||∞
3:      if (ν < ν_max) then return
4:      r ← r − μ, p ← ℳ⁻¹r^(†), ρ ← pᵀr
5:      for k = 0 to k_max do
6:          z ← ℒp, σ ← pᵀz
7:          α ← ρ/σ
8:          r ← r − αz, μ ← r̄, ν ← ||r − μ||∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ, z ← ℳ⁻¹r^(†), ρ^new ← zᵀr
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp, p ← z + βp
17:     end for
18: end procedure
```

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

```
1:  procedure MGPCG(r, x)
2:      r ← r − 𝓛x, μ ← r̄, ν ← ||r − μ||∞
3:      if (ν < ν_max) then return
4:      r ← r − μ, p ← 𝓜^{-1} r^{(†)}, ρ ← p^T r
5:      for k = 0 to k_max do
6:          z ← 𝓛p, σ ← p^T z
7:          α ← ρ/σ
8:          r ← r − αz, μ ← r̄, ν ← ||r − μ||∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ, z ← 𝓜^{-1} r^{(†)}, ρ^new ← z^T r
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp, p ← z + βp
17:     end for
18: end procedure
```

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

```
1:  procedure MGPCG(r, x)
2:      r ← r − ℒx,  μ ← r̄,  ν ← ||r − μ||∞
3:      if (ν < ν_max) then return
4:      r ← r − μ,  p ← ℳ⁻¹r^(†),  [ρ ← pᵀr]
5:      for k = 0 to k_max do
6:          z ← ℒp,  σ ← pᵀz
7:          α ← ρ/σ
8:          r ← r − αz,  μ ← r̄,  ν ← ||r − μ||∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ,  z ← ℳ⁻¹r^(†),  ρ^new ← zᵀr
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp,  p ← z + βp
17:     end for
18: end procedure
```

Reduction Ops (Reductions.h)

LaplaceSolver/LaplaceSolver_0_0

```
#pragma once

#include "Parameters.h"

// Compute the maximum absolute value among the array elements
float Norm(const float (&x)[XDIM][YDIM][ZDIM]);

// Compute the "dot product" between the two arrays
float InnerProduct(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM][ZDIM]);
```

Pointwise Ops (PointwiseOps.cpp)

LaplaceSolver/LaplaceSolver_0_0

```
#include "Reductions.h"

#include <algorithm>

float Norm(const float (&x)[XDIM][YDIM][ZDIM])
{
    float result = 0.;

#pragma omp parallel for reduction(max:result)
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        result = std::max(result, std::abs(x[i][j][k]));

    return result;
}

float InnerProduct(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM][ZDIM])
{
    double result = 0.;

#pragma omp parallel for reduction(+:result)
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        result += (double) x[i][j][k] * (double) y[i][j][k];

    return (float) result;
}
```

Pointwise Ops (PointwiseOps.cpp)

LaplaceSolver/LaplaceSolver_0_0

```
#include "Reductions.h"

#include <algorithm>

float Norm(const float (&x)[XDIM][YDIM][ZDIM])
{
    float result = 0.;

#pragma omp parallel for reduction(max:result)
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        result = std::max(result, std::abs(x[i][j][k]));

    return result;
}

float InnerProduct(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM][ZDIM])
{
    double result = 0.;

#pragma omp parallel for reduction(+:result)
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        result += (double) x[i][j][k] * (double) y[i][j][k];

    return (float) result;
}
```

Test case: Preconditioned Conjugate Gradients

$\mathcal{L}\mathbf{x} = \mathbf{f}$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

Implemented as
rho = InnerProduct(p, r)

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: $\ldots - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\ldots \rho^{\text{new}}/\rho$
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Test case: Preconditioned Conjugate Gradients

$$\mathcal{L}\mathbf{x} = \mathbf{f}$$

Kernels

- Multiply()
- Saxpy()
- Subtract()
- Copy()
- Inner_Product()
- Norm()

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\beta \leftarrow \rho^{\text{new}}/\rho$
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Development Plan

Design

- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

- Implement a prototype
- Organize code into reusable kernels

Accelerate

- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels

Test case: Preconditioned Conjugate Gradients

```
1:  procedure MGPCG(r, x)
2:      r ← r − 𝓛x, μ ← r̄, ν ← ‖r − μ‖∞
3:      if (ν < ν_max) then return
4:      r ← r − μ, p ← 𝓜^{-1} r^{(†)}, ρ ← p^T r
5:      for k = 0 to k_max do
6:          z ← 𝓛p, σ ← p^T z
7:          α ← ρ/σ
8:          r ← r − αz, μ ← r̄, ν ← ‖r − μ‖∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ, z ← 𝓜^{-1} r^{(†)}, ρ^new ← z^T r
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp, p ← z + βp
17:     end for
18: end procedure
```

Test case: Preconditioned Conjugate Gradients

```
1:  procedure MGPCG(r, x)
2:      r ← r − 𝓛x, μ ← r̄, ν ← ||r − μ||_∞
3:      if (ν < ν_max) then return
4:      r ← r − μ, p ← 𝓜^{-1} r^{(†)}, ρ ← p^T r
5:      for k = 0 to k_max do
6:          z ← 𝓛p, σ ← p^T z
7:          α ← ρ/σ
8:          r ← r − αz, μ ← r̄, ν ← ||r − μ||_∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ, z ← 𝓜^{-1} r^{(†)}, ρ^new ← z^T r
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp, p ← z + βp
17:     end for
18: end procedure
```

Test case: Preconditioned Conjugate Gradients

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\beta \leftarrow \rho^{\text{new}}/\rho$
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Test case: Preconditioned Conjugate Gradients

1: **procedure** MGPCG($\mathbf{r}, \mathbf{x}$)
2: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: **if** $(\nu < \nu_{\max})$ **then return**
4: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: **for** $k = 0$ **to** k_{max} **do**
6: $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\alpha \leftarrow \rho/\sigma$
8: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: **if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: **return**
12: **end if**
13: $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\beta \leftarrow \rho^{\text{new}}/\rho$
15: $\rho \leftarrow \rho^{\text{new}}$
16: $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: **end for**
18: **end procedure**

Test case: Preconditioned Conjugate Gradients

```
1:  procedure MGPCG(r, x)
2:      r ← r − 𝓛x,  μ ← r̄,  ν ← ||r − μ||∞
3:      if (ν < ν_max) then return
4:      r ← r − μ,  p ← 𝓜^{-1} r^{(†)},  ρ ← p^T r
5:      for k = 0 to k_max do
6:          z ← 𝓛p,  σ ← p^T z
7:          α ← ρ/σ
8:          r ← r − αz,  μ ← r̄,  ν ← ||r − μ||∞
9:          if (ν < ν_max or k = k_max) then
10:             x ← x + αp
11:             return
12:         end if
13:         r ← r − μ,  z ← 𝓜^{-1} r^{(†)},  ρ^new ← z^T r
14:         β ← ρ^new/ρ
15:         ρ ← ρ^new
16:         x ← x + αp,  p ← z + βp
17:     end for
18: end procedure
```

Test case: Preconditioned Conjugate Gradients

1: **procedure** MGPCG($\mathbf{r},\mathbf{x}$)
2: $\quad\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
3: $\quad$**if** $(\nu < \nu_{\max})$ **then return**
4: $\quad\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
5: $\quad$**for** $k = 0$ **to** k_{max} **do**
6: $\quad\quad\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
7: $\quad\quad\alpha \leftarrow \rho/\sigma$
8: $\quad\quad\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
9: $\quad\quad$**if** $(\nu < \nu_{\max}$ **or** $k = k_{\max})$ **then**
10: $\quad\quad\quad\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
11: $\quad\quad\quad$**return**
12: $\quad\quad$**end if**
13: $\quad\quad\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
14: $\quad\quad\beta \leftarrow \rho^{\text{new}}/\rho$
15: $\quad\quad\rho \leftarrow \rho^{\text{new}}$
16: $\quad\quad\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
17: $\quad$**end for**
18: **end procedure**

Development Plan

Design

- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

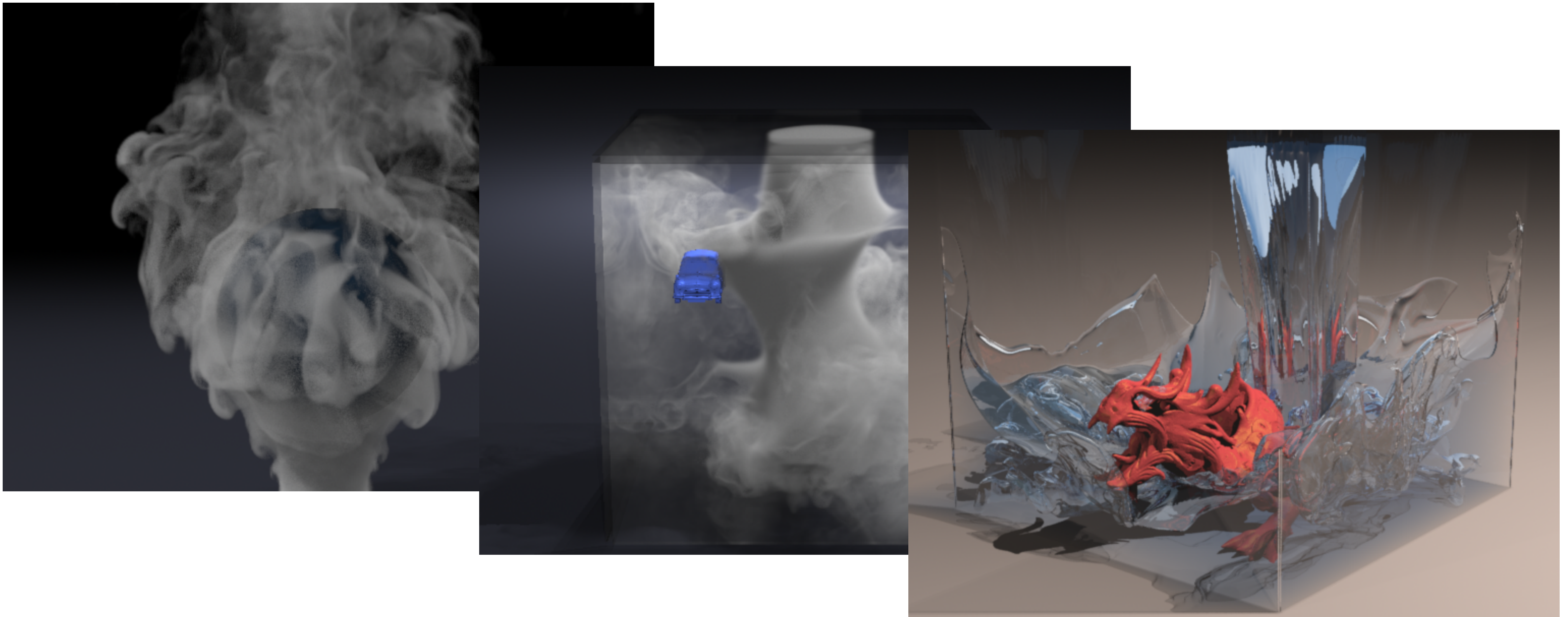
- Implement a prototype
- Organize code into reusable kernels

Accelerate

- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels

Coming attractions

- We will see implementations of the various "kernels" that showed up in pseudo-code
- We will address challenges to parallel performance for each of them
- We will investigate the impact of "merging" kernels, when possible (your next homework)
- You will be given a "framework" that assembles those kernels into a solver (without having to worry too much about the theory) [next lecture]



Stencil computations in the context of solving sparse linear systems
(motivated by computational physics and graphics)