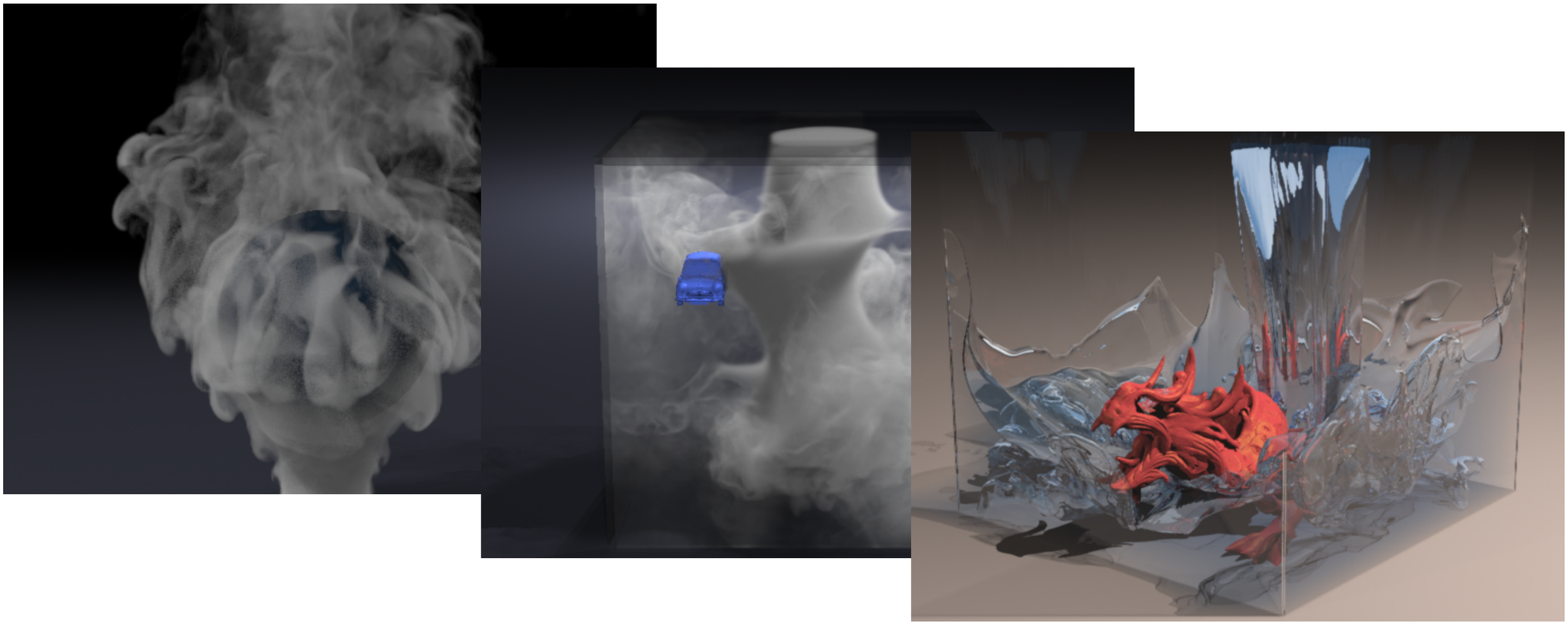


Stencil computations in the context of solving sparse linear systems (motivated by computational physics and graphics)

Introduction

- Linear systems of equations are ubiquitous in engineering
- Their solution is often the bottleneck for entire applications

Introduction

- Linear systems of equations are ubiquitous in engineering
- Their solution is often the bottleneck for entire applications
- NxN systems Ax=b arising from physics have special properties:
  - Almost always : Sparse - O(N) nonzero entries
  - Very often : Symmetric
  - Often enough : Positive definite, diagonally dominant, etc.
  - Dense matrix complexity ~$O(N^{2.38})$ does not apply

Introduction

- Linear systems of equations are ubiquitous in engineering
- Their solution is often the bottleneck for entire applications
- NxN systems Ax=b arising from physics have special properties:
  - Almost always : Sparse - O(N) nonzero entries
  - Very often : Symmetric
  - Often enough : Positive definite, diagonally dominant, etc.
  - Dense matrix complexity ~$O(N^{2.38})$ does not apply
- For special matrices, solvers with cost as low as O(N) exist
  - The "hidden constant" becomes important, and can be lowered via parallelism
  - Efficient algorithms need to be very frugal with storage

Introduction

Example : The 3D Poisson equation

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

Benchmark launcher (main.cpp)

LaplacianStencil_0_10

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];

    float *uRaw = new float [XDIM*YDIM*ZDIM];
    float *LuRaw = new float [XDIM*YDIM*ZDIM];
    array_t u = reinterpret_cast<array_t>(*uRaw);
    array_t Lu = reinterpret_cast<array_t>(*LuRaw);

    Timer timer;

    for(int test = 1; test <= 10; test++)
    {
        std::cout << "Running test iteration " << std::setw(2) << test << " ";
        timer.Start();
        ComputeLaplacian(u, Lu);
        timer.Stop("Elapsed time : ");
    }

    return 0;
}
```

Kernel header (Laplacian.h)

LaplacianStencil_0_10

```
#pragma once

#define XDIM 512
#define YDIM 512
#define ZDIM 512

void ComputeLaplacian(const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM]);
```

Kernel Body (Laplacian.cpp)

LaplacianStencil_0_10

```
#include "Laplacian.h"

void ComputeLaplacian(const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[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++)
        Lu[i][j][k] =
        -6 * u[i][j][k]
        + u[i+1][j][k]
        + u[i-1][j][k]
        + u[i][j+1][k]
        + u[i][j-1][k]
        + u[i][j][k+1]
        + u[i][j][k-1];

}
```

Kernel Body (Laplacian.cpp)

LaplacianStencil_0_10

```
#include "Laplacian.h"

void ComputeLaplacian(const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[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++)
        Lu[i][j][k] =
        -6 * u[i][j][k]
        + u[i+1][j][k]
        + u[i-1][j][k]
        + u[i][j+1][k]
        + u[i][j-1][k]
        + u[i][j][k+1]
        + u[i][j][k-1];

}
```

Execution:

```
Running test iteration  1 [Elapsed time : 76.9873ms]
Running test iteration  2 [Elapsed time : 13.7364ms]
Running test iteration  3 [Elapsed time : 13.7121ms]
Running test iteration  4 [Elapsed time : 13.6728ms]
Running test iteration  5 [Elapsed time : 13.6913ms]
Running test iteration  6 [Elapsed time : 14.7445ms]
Running test iteration  7 [Elapsed time : 13.6666ms]
Running test iteration  8 [Elapsed time : 13.6243ms]
Running test iteration  9 [Elapsed time : 13.6784ms]
Running test iteration 10 [Elapsed time : 13.5126ms]
```

Introduction

Example : The 3D Poisson equation

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

What about x & b?
How are they stored?

Introduction

Example : The 3D Poisson equation

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

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

What about x & b?
How are they stored?

Introduction

Example : The 3D Poisson equation

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

What about x & b?
How are they stored?

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

Introduction

Example : The 3D Poisson equation

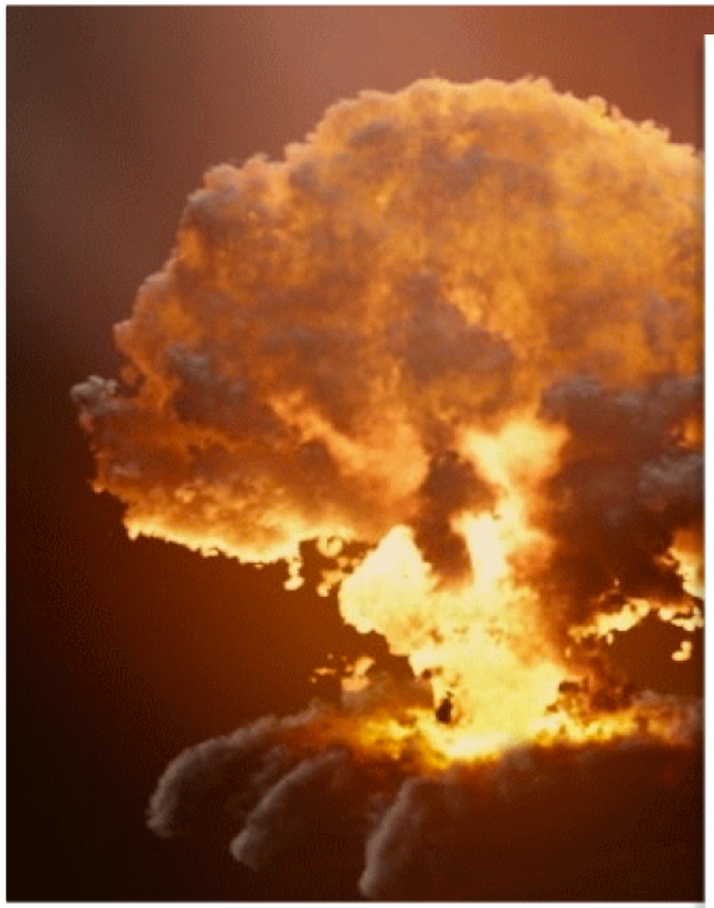
$$\begin{pmatrix}
-6 & 1 & & & 1 & & & 1 & & & \\
1 & -6 & 1 & & & 1 & & & \ddots & & \\
 & 1 & -6 & 1 & & & \ddots & & & \ddots & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & 1 \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & \\
 & & \ddots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

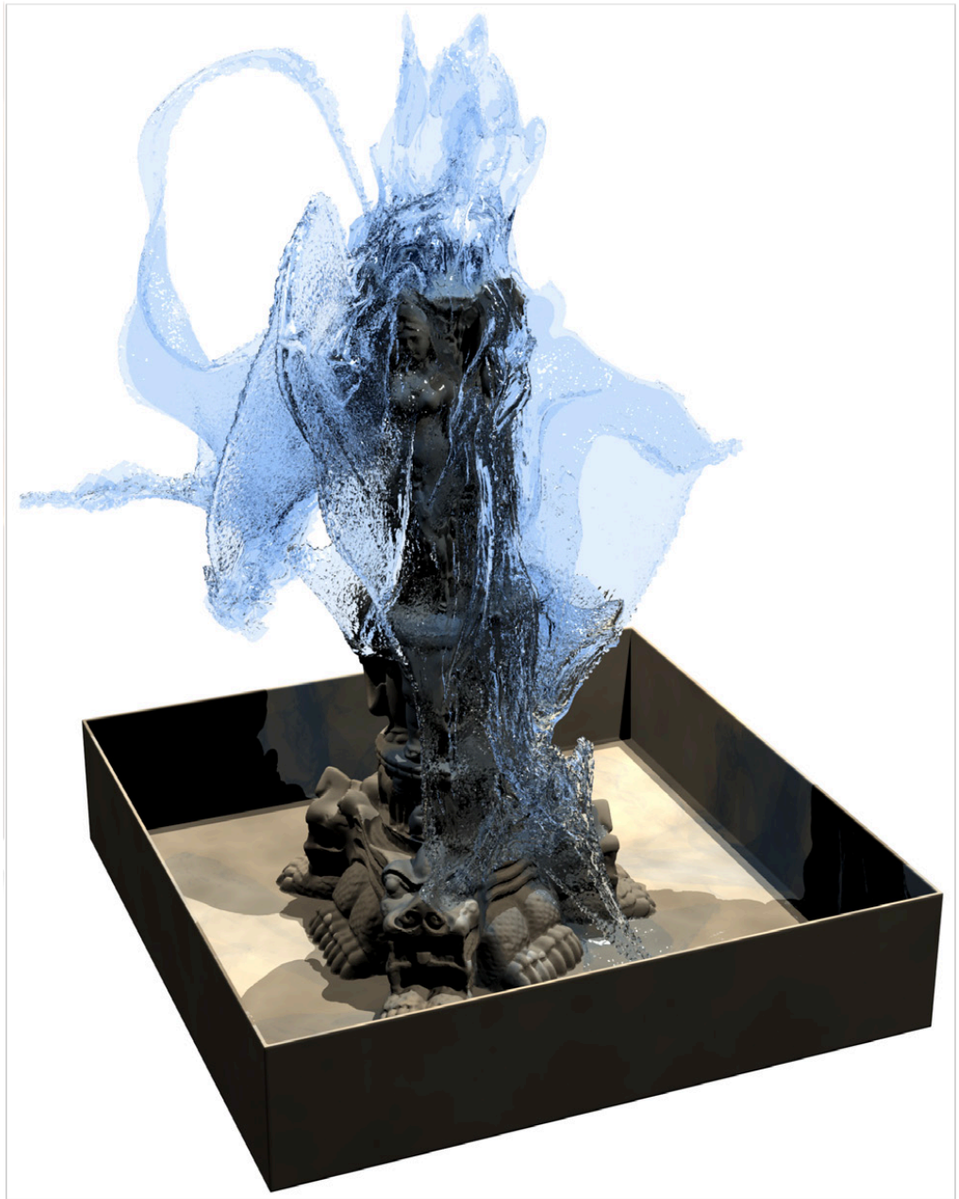
*Computing b = L*x is equivalent to executing*
ComputeLaplacian(u,Lu)

Applications

Rasmussen et al, Smoke Simulation for Large Scale Phenomena (SIGGRAPH 2003)

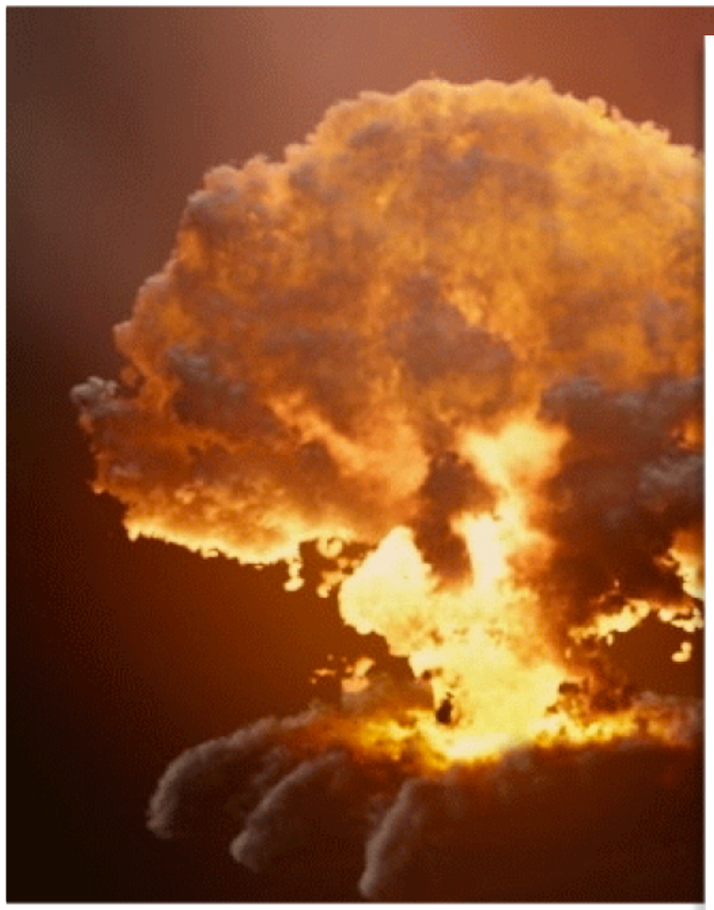
Applications





Nielsen et al, Out-Of-Core and Compressed Level Set Methods (ACM TOG 2007)

Applications



Horvath & Geiger, Directable, high-resolution simulation of fire on the GPU (ACM SIGGRAPH 2009)

Applications

Governing equation and boundary conditions

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

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

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

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

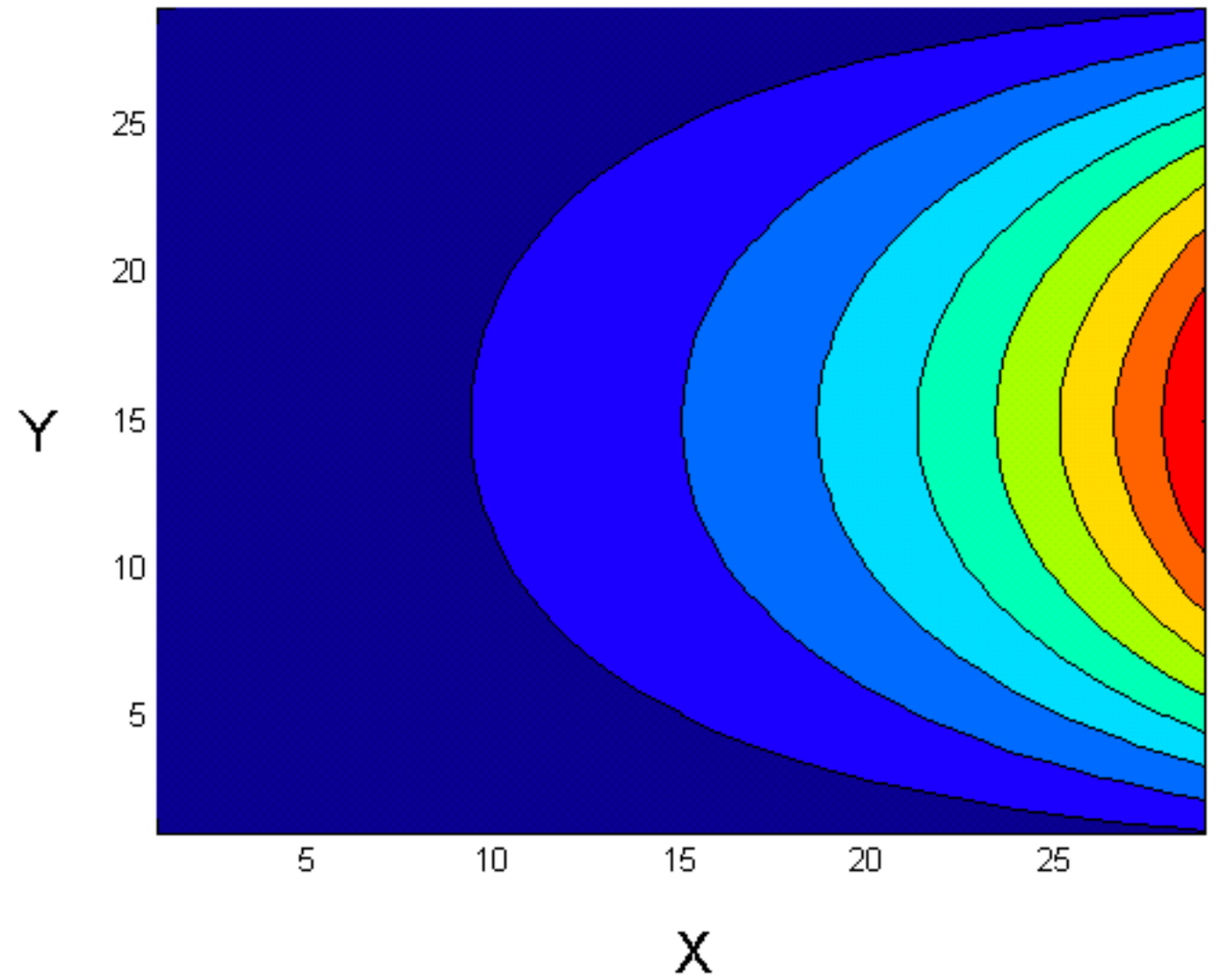
Finite difference scheme

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

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

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

Temperature profile in a rectangular plate



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 $\boldsymbol{Ax=b}$ 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?

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

- The system $\boldsymbol{Ax=b}$ 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

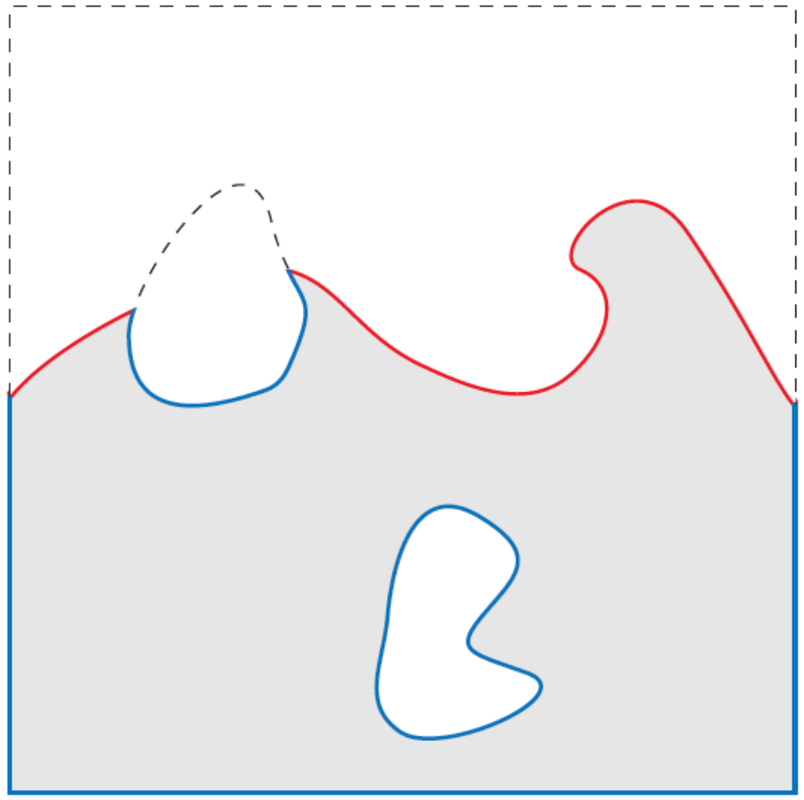
What kind of accuracy do we need?

- 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?

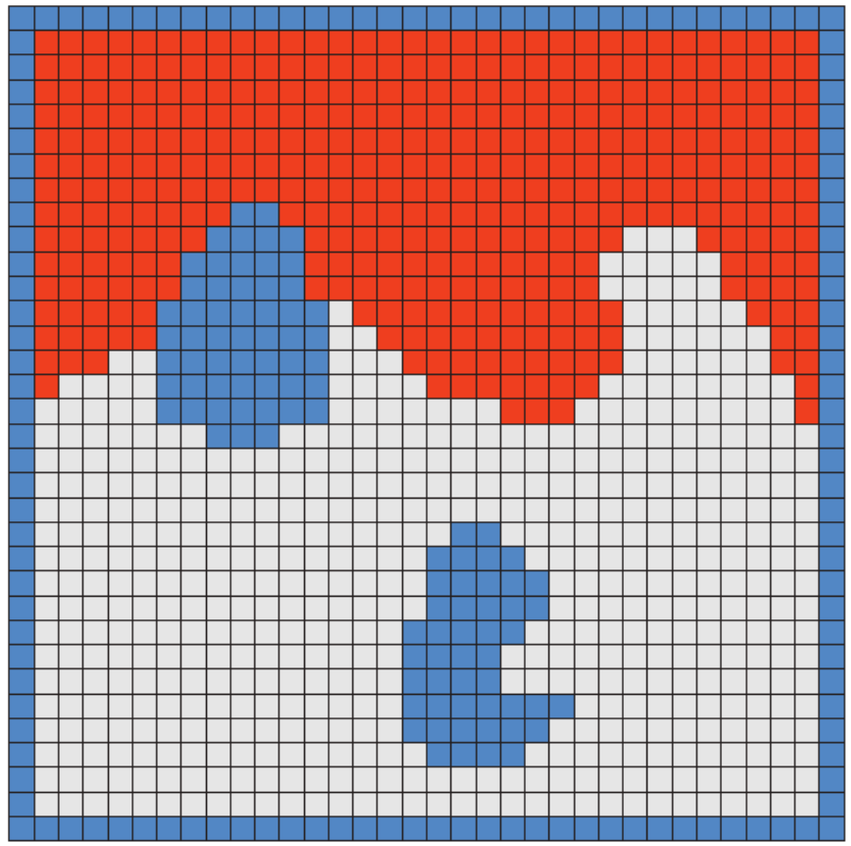
What is the *real* underlying problem we care about?

- 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

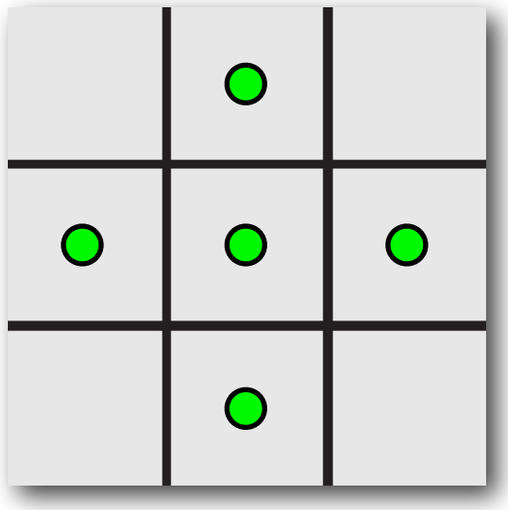


$$\Delta \mathbf{x} = \mathbf{b}$$



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

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



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?

$$\mathbf{Ax} = \mathbf{b}$$

"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

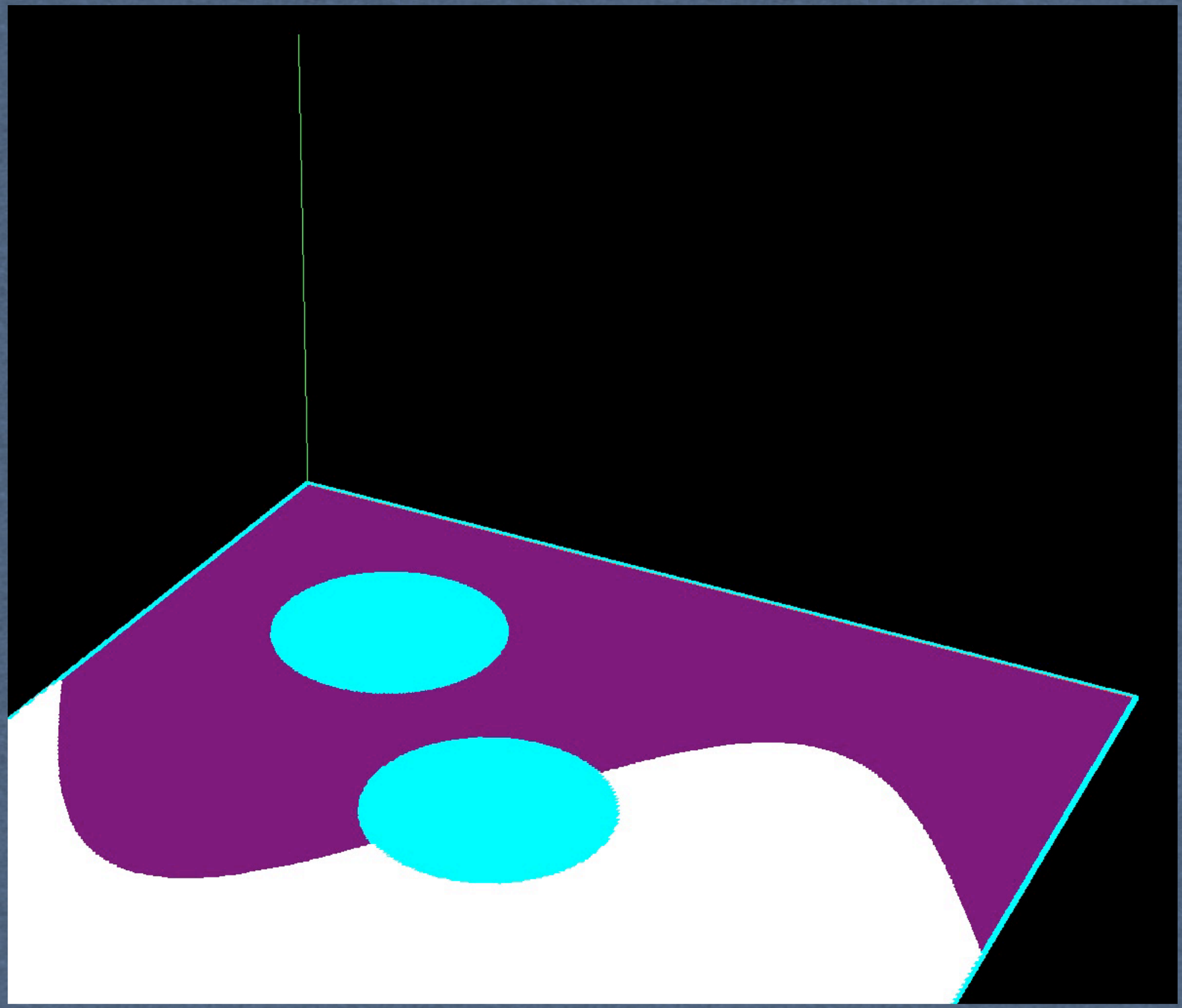
- Define your objectives
- Choose a parallel-friendly theoretical formulation
- Set performance expectations
- Choose a promising algorithm

Implement

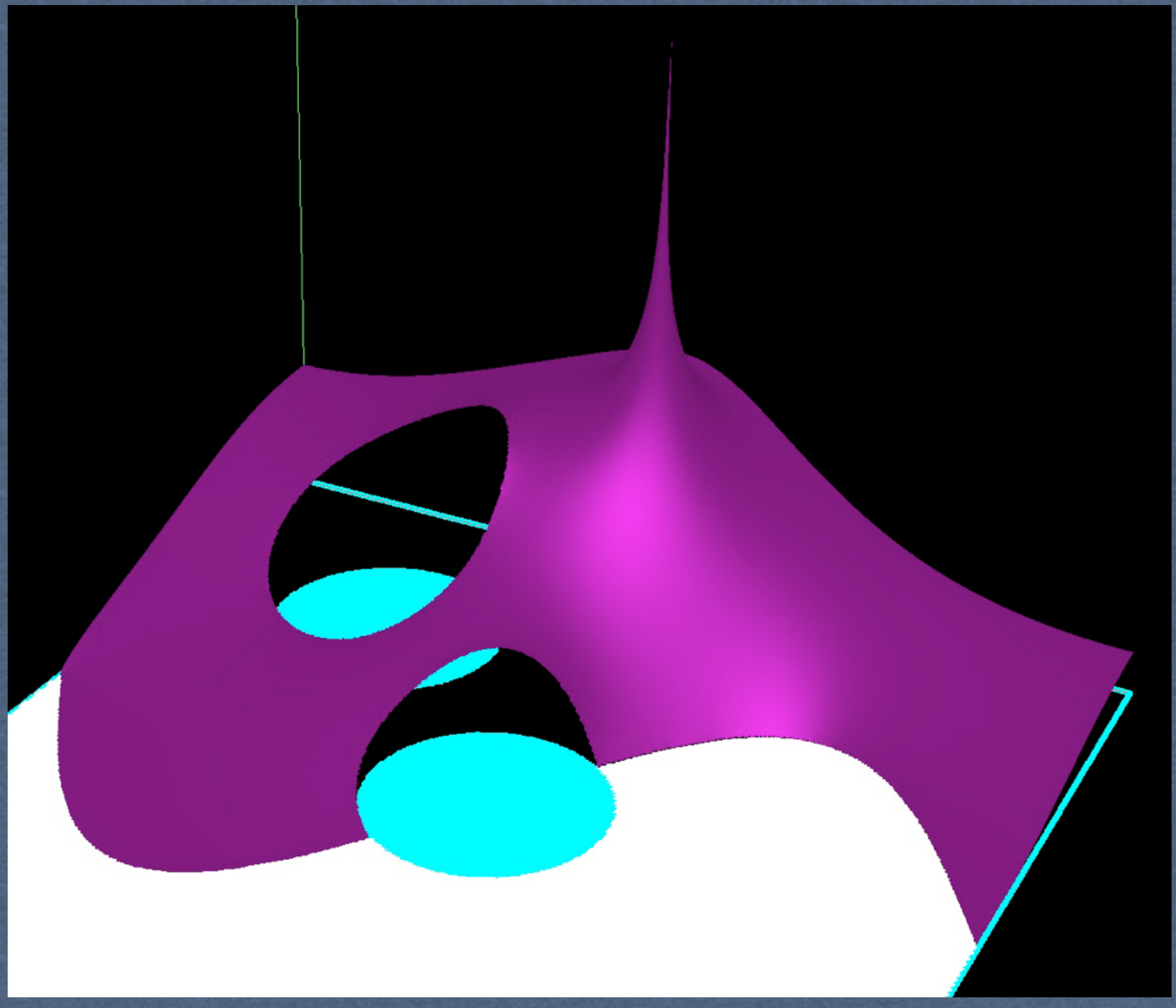
- Implement a prototype
- Organize code into reusable kernels

Accelerate

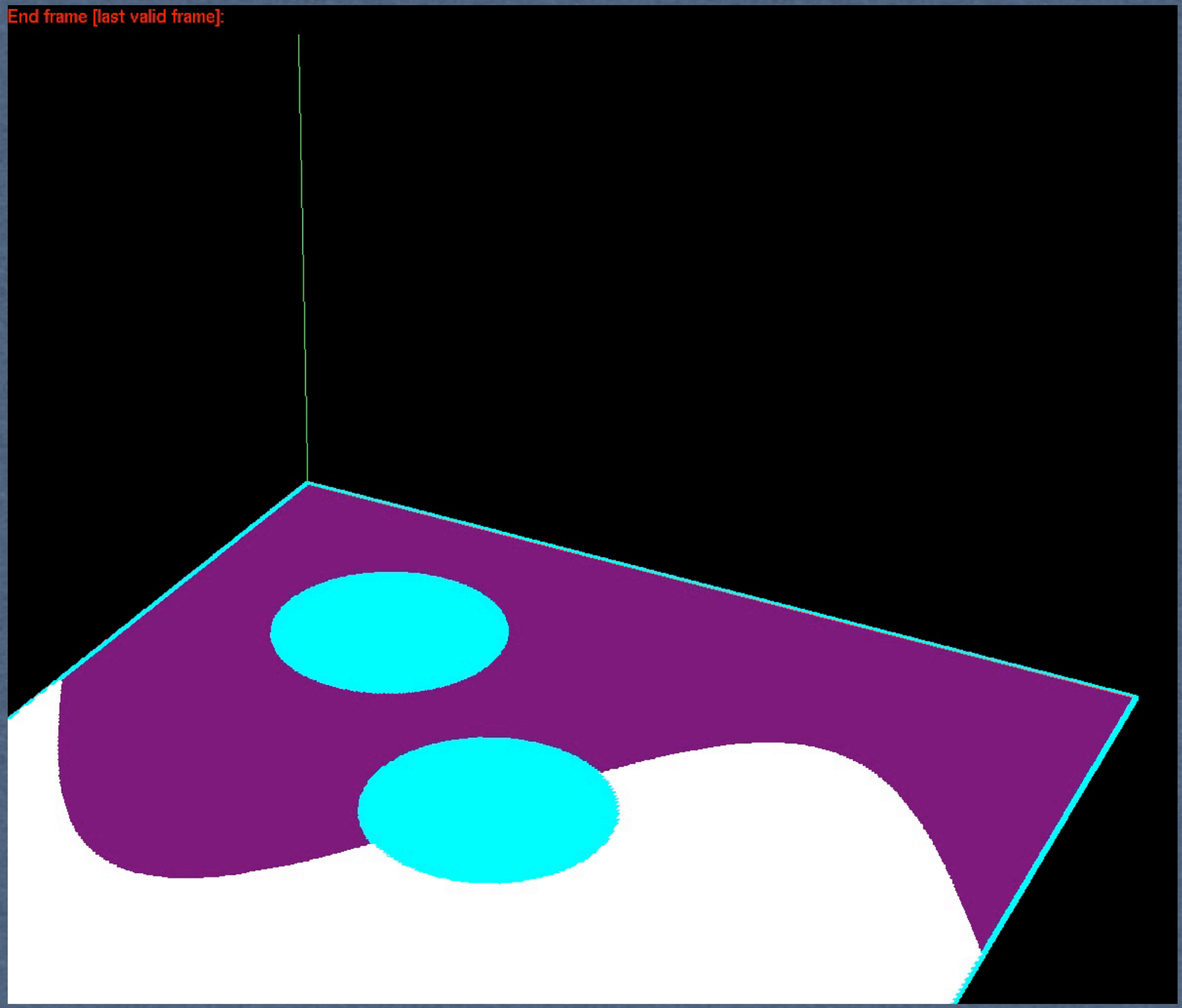
- Reorder/combine/pipeline operations
- Reduce resource utilization (try harder ...)
- Parallelize component kernels



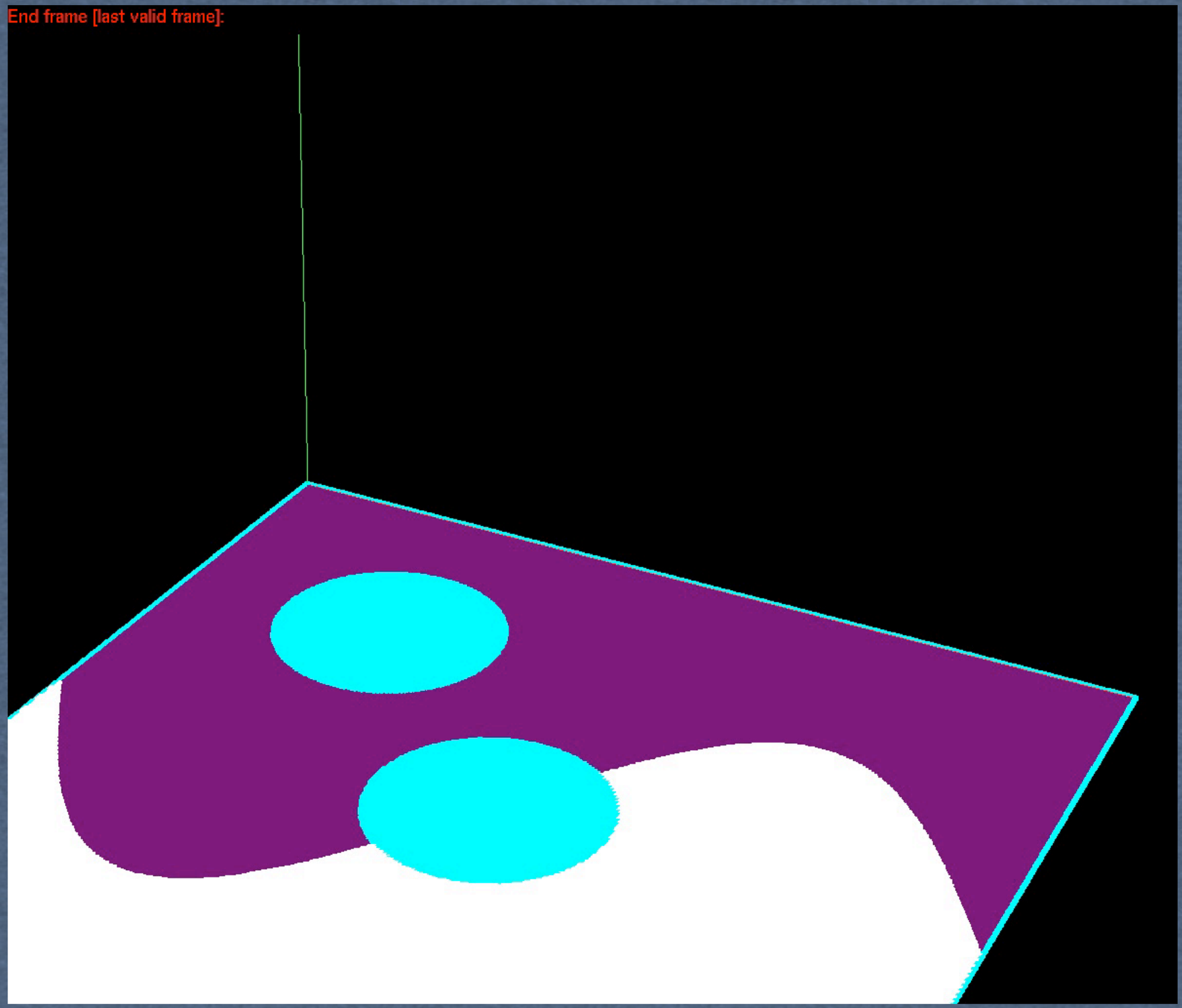
Sample 2D domain (512x512 resolution)



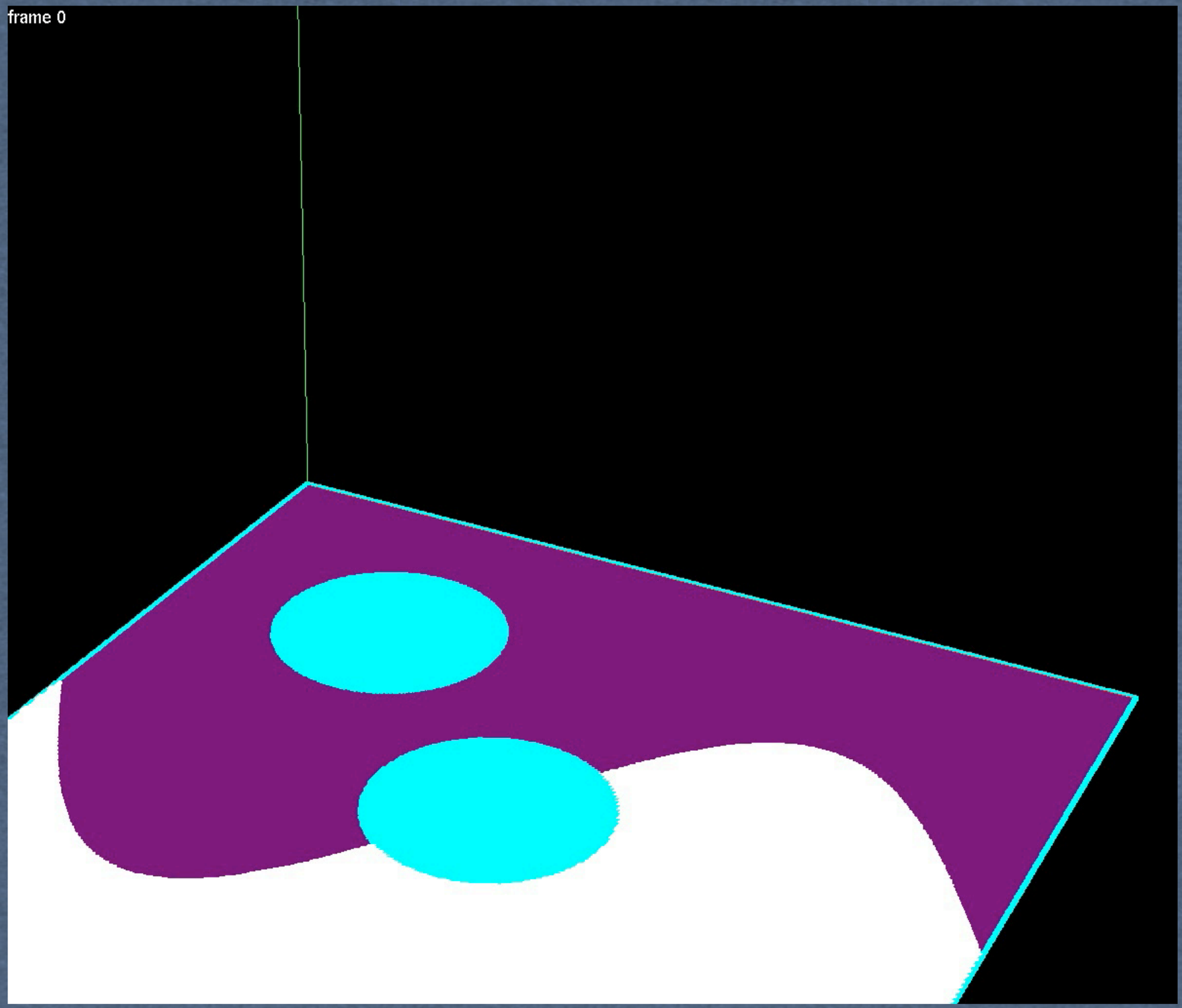
Exact solution



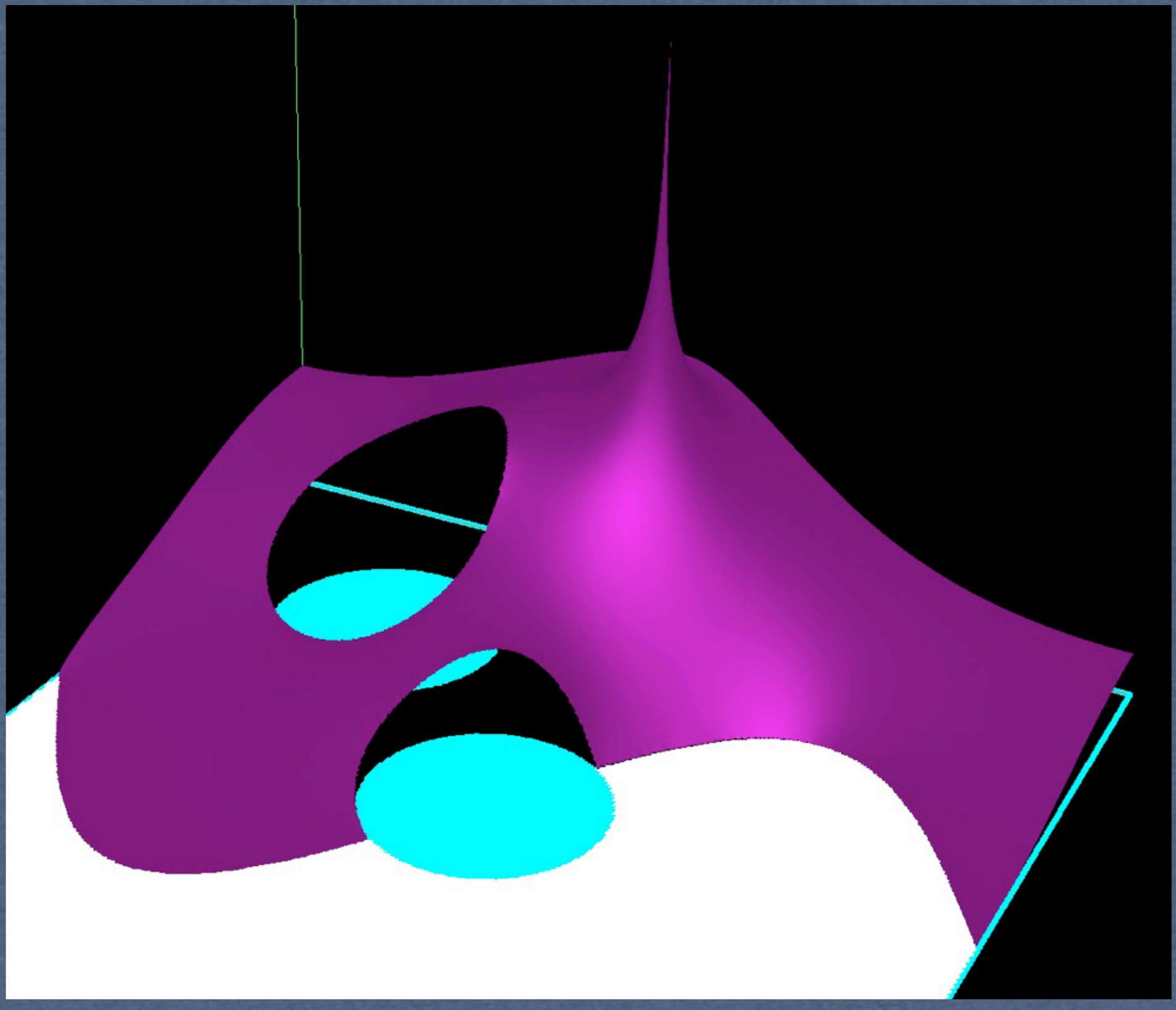
Conjugate Gradients (w/o preconditioning)



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: $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
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

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

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: $\quad\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
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: $\quad\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
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: $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
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

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

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

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

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**

Kernels

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

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($\mathbf{r}, \mathbf{x}$)
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

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

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

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

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

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

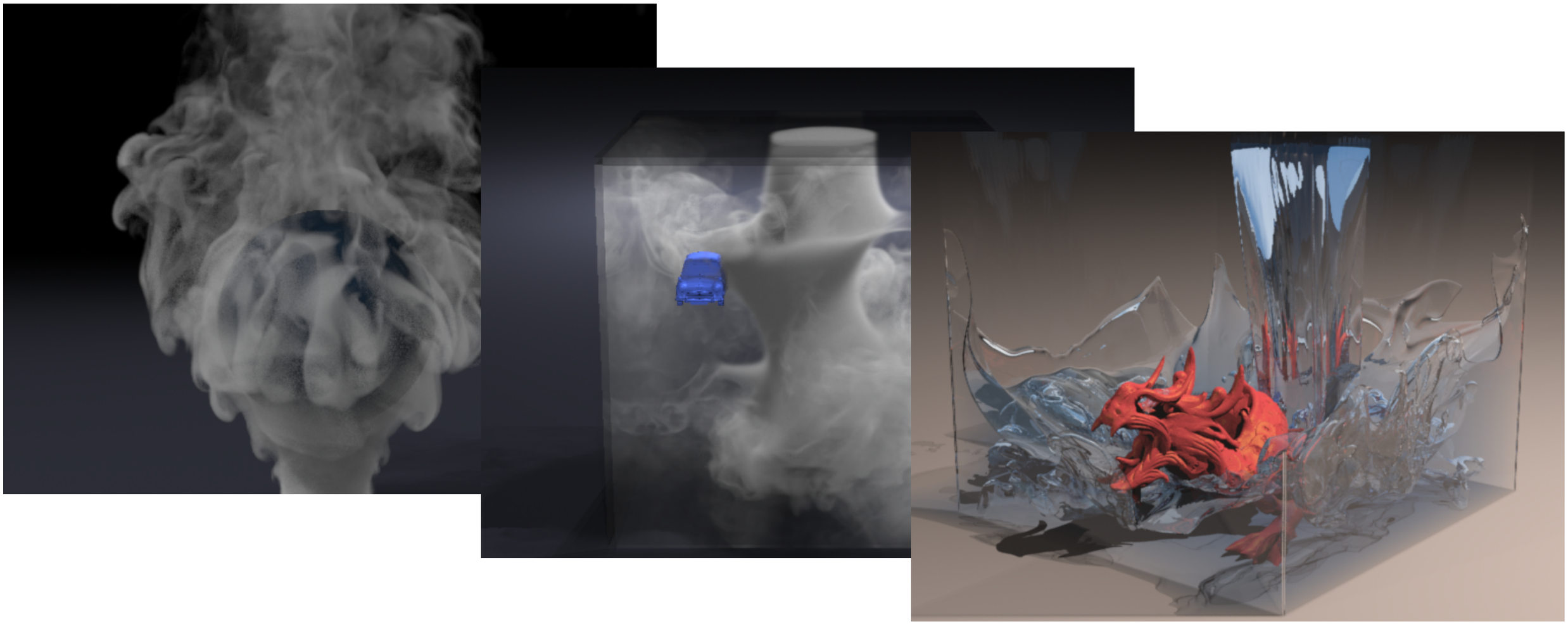
- Implement a prototype
- Organize code into reusable kernels

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
- You will be given a "framework" that assembles those kernels into a solver (without having to worry too much about the theory)



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