Ein adaptives, paralleles Lattice-Boltzmann Verfahren zur Simulation komplexen Wirbelschleppentransports hinter bewegten Geometrien

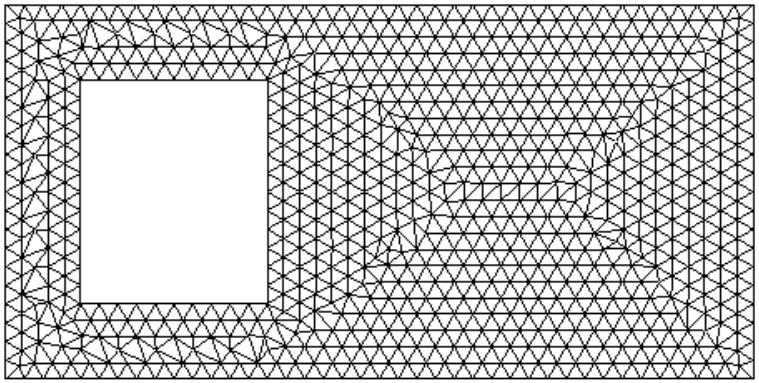
Ralf Deiterding

Deutsches Zentrum für Luft- und Raumfahrt
Bunsenstr. 10, Göttingen, Germany
E-mail: ralf.deiterding@dlr.de

September 18, 2014

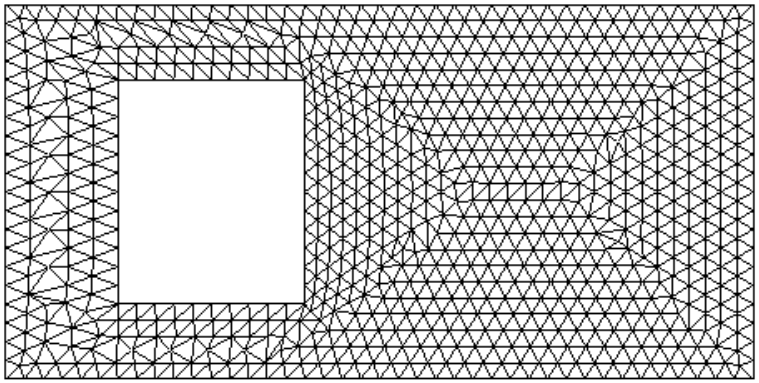
Meshing techniques for FSI

Unstructured:



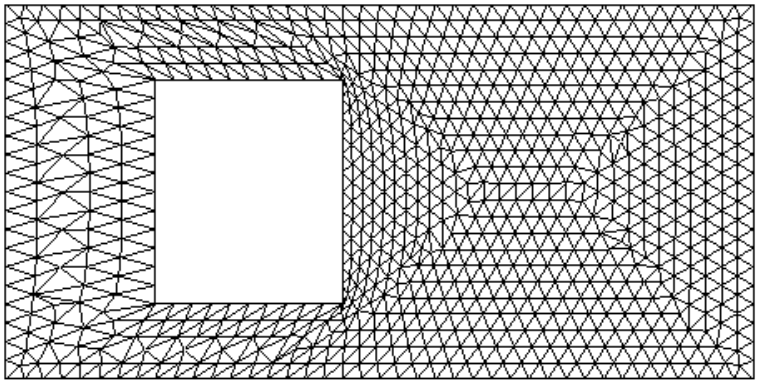
Meshing techniques for FSI

Unstructured:



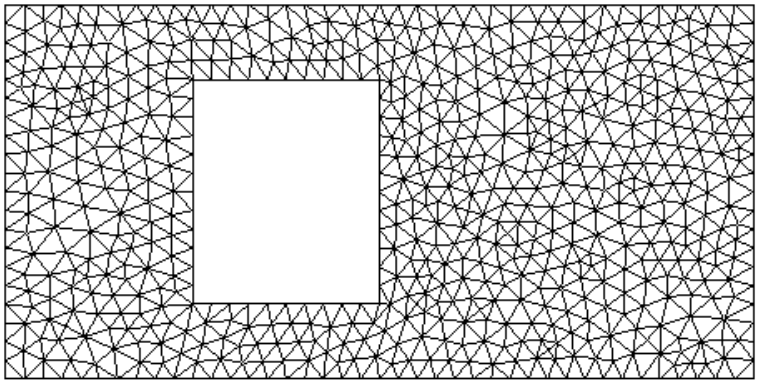
Meshing techniques for FSI

Unstructured:



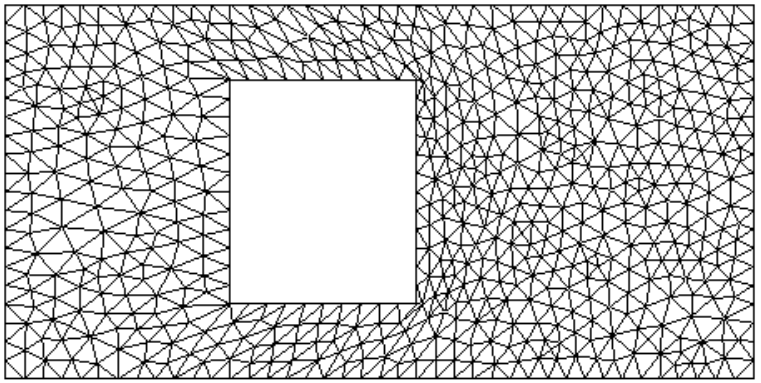
Meshing techniques for FSI

Unstructured:



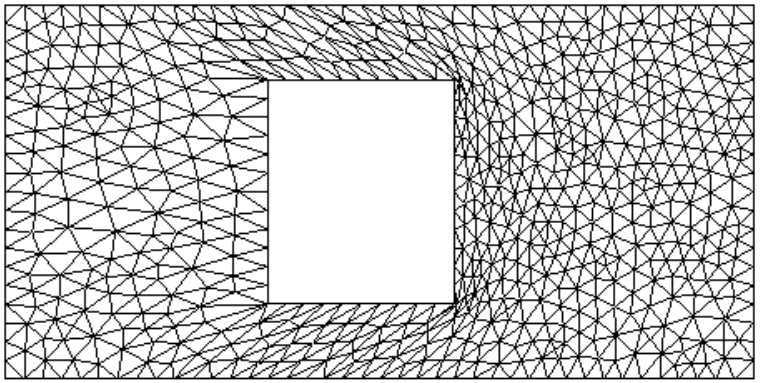
Meshing techniques for FSI

Unstructured:



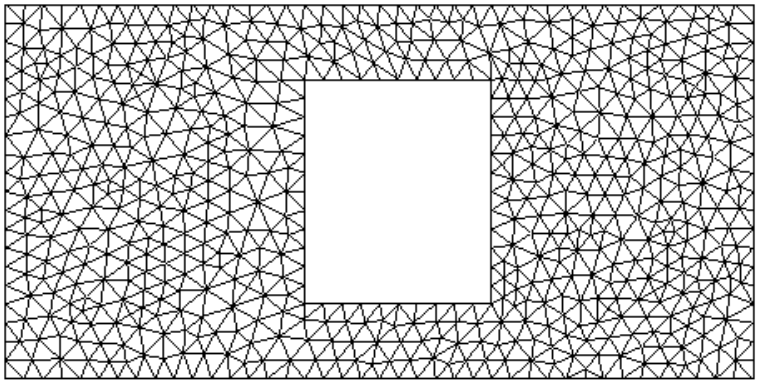
Meshing techniques for FSI

Unstructured:



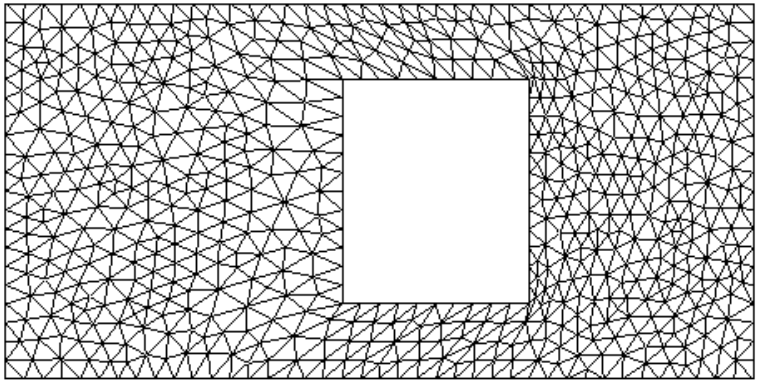
Meshing techniques for FSI

Unstructured:



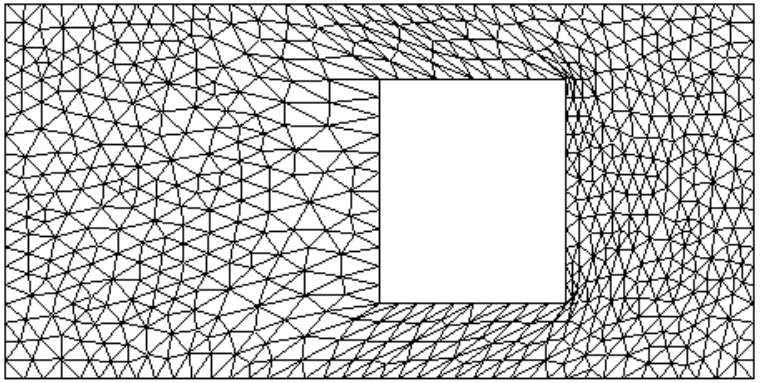
Meshing techniques for FSI

Unstructured:



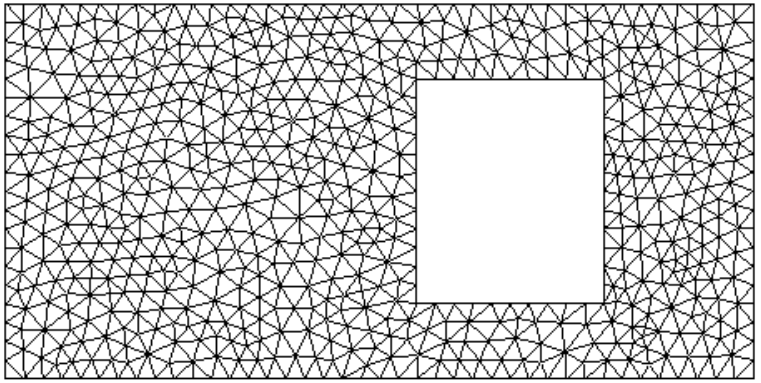
Meshing techniques for FSI

Unstructured:

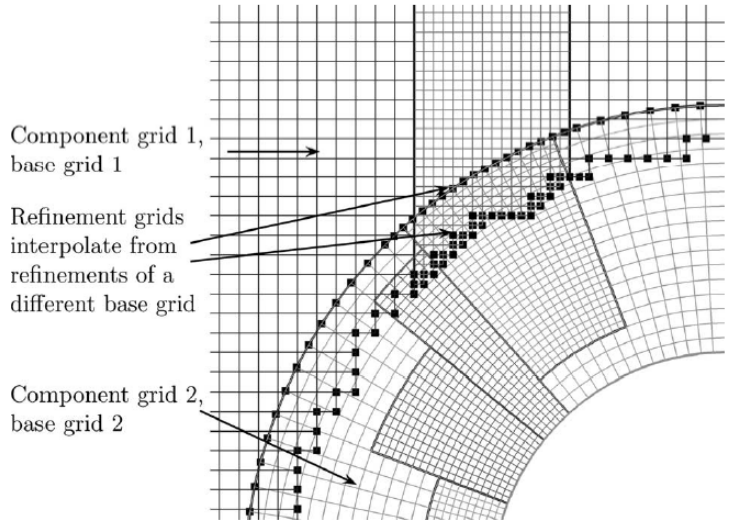


Meshing techniques for FSI

Unstructured:



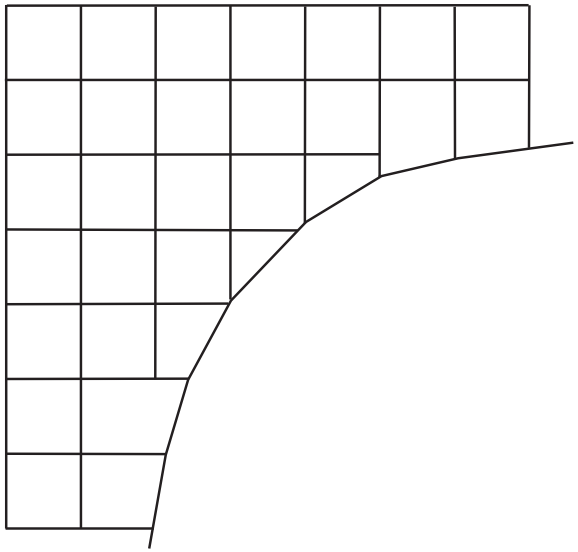
Structured:



Chimera, overlapping adaptive meshes

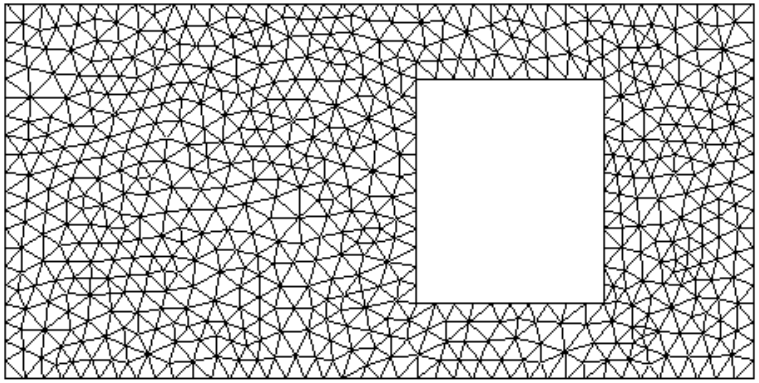
Meshing techniques for FSI

Cartesian:

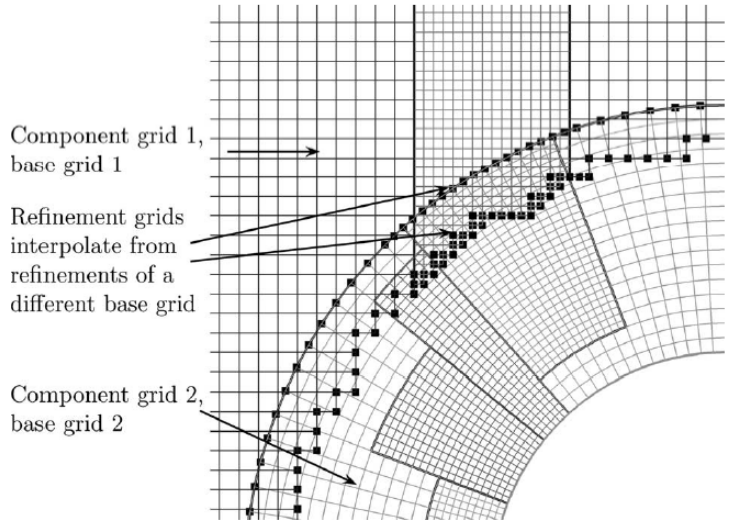


- Methods that represent the boundary sharply
- Methods that diffuse the boundary

Unstructured:



Structured:



Chimera, overlapping adaptive meshes

Outline

Approximation of Boltzmann equation

Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f / L \ll 1$, where l_f is replaced with Δx

Approximation of Boltzmann equation

Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f / L \ll 1$, where l_f is replaced with Δx

Equation is approximated with a splitting approach.

1.) Transport step solves $\partial_t f_\alpha + \mathbf{e}_\alpha \cdot \nabla f_\alpha = 0$

Operator: $\mathcal{T}$: $\tilde{f}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t)$

Approximation of Boltzmann equation

Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f / L \ll 1$, where l_f is replaced with Δx

Equation is approximated with a splitting approach.

1.) Transport step solves $\partial_t f_\alpha + \mathbf{e}_\alpha \cdot \nabla f_\alpha = 0$

Operator: $\mathcal{T}$: $\tilde{f}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t)$

$$\rho(\mathbf{x}, t) = \sum_{\alpha=0}^{8} f_\alpha(\mathbf{x}, t), \quad \rho(\mathbf{x}, t) u_i(\mathbf{x}, t) = \sum_{\alpha=0}^{8} \mathbf{e}_{\alpha i} f_\alpha(\mathbf{x}, t)$$

Approximation of Boltzmann equation

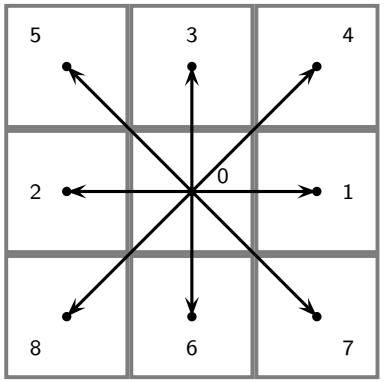
Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f / L \ll 1$, where l_f is replaced with Δx

Equation is approximated with a splitting approach.

1.) Transport step solves $\partial_t f_\alpha + \mathbf{e}_\alpha \cdot \nabla f_\alpha = 0$

Operator: $\mathcal{T}$: $\tilde{f}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t)$

$$\rho(\mathbf{x}, t) = \sum_{\alpha=0}^{8} f_\alpha(\mathbf{x}, t), \quad \rho(\mathbf{x}, t) u_i(\mathbf{x}, t) = \sum_{\alpha=0}^{8} \mathbf{e}_{\alpha i} f_\alpha(\mathbf{x}, t)$$

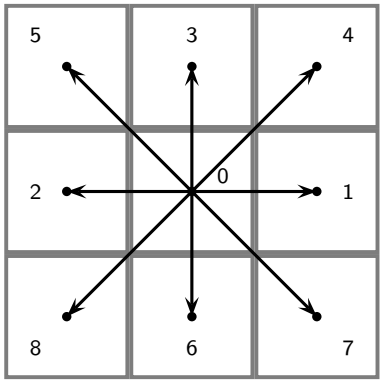


Discrete velocities:
$\mathbf{e}_0 = (0,0), \mathbf{e}_1 = (1,0)c, \mathbf{e}_2 = (-1,0)c, \mathbf{e}_3 = (0,1)c, \mathbf{e}_4 = (1,1)c, \ldots$

Approximation of Boltzmann equation

Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f / L \ll 1$, where l_f is replaced with Δx

Equation is approximated with a splitting approach.



1.) Transport step solves $\partial_t f_\alpha + \mathbf{e}_\alpha \cdot \nabla f_\alpha = 0$

Operator: $\mathcal{T}$: $\tilde{f}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t)$

$$\rho(\mathbf{x}, t) = \sum_{\alpha=0}^{8} f_\alpha(\mathbf{x}, t), \quad \rho(\mathbf{x}, t) u_i(\mathbf{x}, t) = \sum_{\alpha=0}^{8} \mathbf{e}_{\alpha i} f_\alpha(\mathbf{x}, t)$$

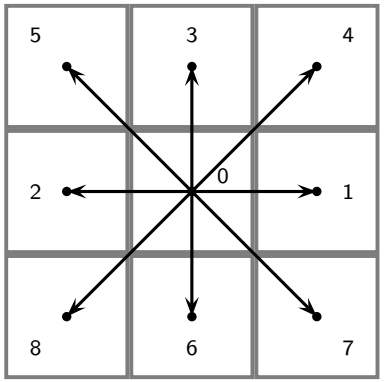
Discrete velocities:
$\mathbf{e}_0 = (0,0), \mathbf{e}_1 = (1,0)c, \mathbf{e}_2 = (-1,0)c, \mathbf{e}_3 = (0,1)c, \mathbf{e}_4 = (1,1)c, \ldots$

$c = \dfrac{\Delta x}{\Delta t}$, Physical speed of sound: $c_s = \dfrac{c}{\sqrt{3}}$

Approximation of Boltzmann equation

Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f/L \ll 1$, where l_f is replaced with Δx

Equation is approximated with a splitting approach.



1.) Transport step solves $\partial_t f_\alpha + \mathbf{e}_\alpha \cdot \nabla f_\alpha = 0$

Operator: $\mathcal{T}$: $\tilde{f}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t)$

$$\rho(\mathbf{x}, t) = \sum_{\alpha=0}^{8} f_\alpha(\mathbf{x}, t), \quad \rho(\mathbf{x}, t) u_i(\mathbf{x}, t) = \sum_{\alpha=0}^{8} \mathbf{e}_{\alpha i} f_\alpha(\mathbf{x}, t)$$

Discrete velocities:
$\mathbf{e}_0 = (0,0), \mathbf{e}_1 = (1,0)c, \mathbf{e}_2 = (-1,0)c, \mathbf{e}_3 = (0,1)c, \mathbf{e}_4 = (1,1)c, \ldots$

$c = \dfrac{\Delta x}{\Delta t}$, Physical speed of sound: $c_s = \dfrac{c}{\sqrt{3}}$

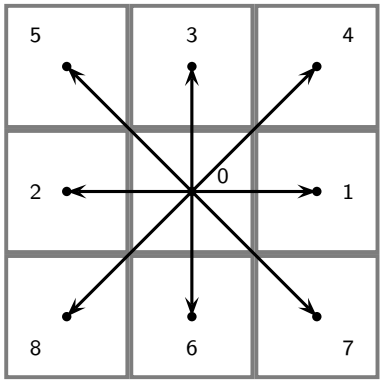
2.) Collision step solves $\partial_t f_\alpha = \omega(f_\alpha^{eq} - f_\alpha)$

Operator $\mathcal{C}$: $f_\alpha(\cdot, t + \Delta t) = \tilde{f}_\alpha(\cdot, t + \Delta t) + \omega \Delta t \left(\tilde{f}_\alpha^{eq}(\cdot, t + \Delta t) - \tilde{f}_\alpha(\cdot, t + \Delta t) \right)$

Approximation of Boltzmann equation

Is based on solving the Boltzmann equation $\partial_t f + \mathbf{u} \cdot \nabla f = \omega(f^{eq} - f)$ with a simplified collision operator. Assumption: $\mathrm{Kn} = l_f/L \ll 1$, where l_f is replaced with Δx

Equation is approximated with a splitting approach.



1.) Transport step solves $\partial_t f_\alpha + \mathbf{e}_\alpha \cdot \nabla f_\alpha = 0$

Operator: $\mathcal{T}$: $\tilde{f}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t)$

$$\rho(\mathbf{x}, t) = \sum_{\alpha=0}^{8} f_\alpha(\mathbf{x}, t), \quad \rho(\mathbf{x}, t) u_i(\mathbf{x}, t) = \sum_{\alpha=0}^{8} \mathbf{e}_{\alpha i} f_\alpha(\mathbf{x}, t)$$

Discrete velocities:
$\mathbf{e}_0 = (0,0), \mathbf{e}_1 = (1,0)c, \mathbf{e}_2 = (-1,0)c, \mathbf{e}_3 = (0,1)c, \mathbf{e}_4 = (1,1)c, \ldots$

$c = \dfrac{\Delta x}{\Delta t}$, Physical speed of sound: $c_s = \dfrac{c}{\sqrt{3}}$

2.) Collision step solves $\partial_t f_\alpha = \omega(f_\alpha^{eq} - f_\alpha)$

Operator $\mathcal{C}$: $f_\alpha(\cdot, t + \Delta t) = \tilde{f}_\alpha(\cdot, t + \Delta t) + \omega \Delta t \left(\tilde{f}_\alpha^{eq}(\cdot, t + \Delta t) - \tilde{f}_\alpha(\cdot, t + \Delta t) \right)$

with equilibrium function

$$f_\alpha^{eq}(\rho, \mathbf{u}) = \rho t_\alpha \left[1 + \frac{3 \mathbf{e}_\alpha \mathbf{u}}{c^2} + \frac{9(\mathbf{e}_\alpha \mathbf{u})^2}{2c^4} - \frac{3\mathbf{u}^2}{2c^2} \right]$$

with $t_\alpha = \frac{1}{9} \left\{ 4, 1, 1, 1, \frac{1}{4}, \frac{1}{4}, 1, \frac{1}{4}, \frac{1}{4} \right\}$. Pressure $\delta p = \sum_\alpha f_\alpha^{eq} c_s^2 = (\rho - \rho_0) c_s^2$.

Relation to Navier-Stokes equations

Inserting a Chapman-Enskog expansion, that is,

$$f_\alpha = f_\alpha(0) + \epsilon f_\alpha(1) + \epsilon^2 f_\alpha(2) + ...$$

and using

$$\frac{\partial}{\partial t} = \epsilon \frac{\partial}{\partial t_1} + \epsilon^2 \frac{\partial}{\partial t_2} + ..., \qquad \nabla = \epsilon \nabla_1 + \epsilon^2 \nabla_2 + ...$$

into the LBM and summing over α one can show that the continuity and moment equations are recoverd to $O(\epsilon^2)$ [Hou et al., 1996]

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}$$

Relation to Navier-Stokes equations

Inserting a Chapman-Enskog expansion, that is,

$$f_\alpha = f_\alpha(0) + \epsilon f_\alpha(1) + \epsilon^2 f_\alpha(2) + ...$$

and using

$$\frac{\partial}{\partial t} = \epsilon \frac{\partial}{\partial t_1} + \epsilon^2 \frac{\partial}{\partial t_2} + ..., \qquad \nabla = \epsilon \nabla_1 + \epsilon^2 \nabla_2 + ...$$

into the LBM and summing over α one can show that the continuity and moment equations are recoverd to $O(\epsilon^2)$ [Hou et al., 1996]

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}$$

Kinematic viscosity and collision time are connected by

$$\nu = \frac{1}{3}\left(\tau - \frac{1}{2}\right) c \Delta x$$

from which one gets with $\sqrt{3} c_s = \frac{\Delta x}{\Delta t}$ [Hähnel, 2004]

$$\omega = \tau^{-1} = \frac{c_s^2 \Delta t}{\nu + \Delta t c_s^2 / 2}$$

High Reynolds numbers

Pursue a large-eddy simulation approach with $\bar{f}_\alpha$ and $\bar{f}_\alpha^{eq}$, i.e.

1.) $\tilde{\bar{f}}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = \bar{f}_\alpha(\mathbf{x}, t)$

2.) $\bar{f}_\alpha(\cdot, t + \Delta t) = \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) + \frac{1}{\tau^\star} \Delta t \left(\tilde{\bar{f}}_\alpha^{eq}(\cdot, t + \Delta t) - \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) \right)$

High Reynolds numbers

Pursue a large-eddy simulation approach with $\bar{f}_\alpha$ and $\bar{f}_\alpha^{eq}$, i.e.

1.) $\tilde{\bar{f}}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = \bar{f}_\alpha(\mathbf{x}, t)$

2.) $\bar{f}_\alpha(\cdot, t + \Delta t) = \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) + \frac{1}{\tau^\star} \Delta t \left(\tilde{\bar{f}}_\alpha^{eq}(\cdot, t + \Delta t) - \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) \right)$

Effective viscosity: $\nu^\star = \nu + \nu_t = \frac{1}{3} \left(\tau^\star - \frac{1}{2} \right) c \Delta x \quad \text{with} \quad \tau^\star = \tau + \tau_t$

High Reynolds numbers

Pursue a large-eddy simulation approach with $\bar{f}_\alpha$ and $\bar{f}_\alpha^{eq}$, i.e.

1.) $\tilde{\bar{f}}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = \bar{f}_\alpha(\mathbf{x}, t)$

2.) $\bar{f}_\alpha(\cdot, t + \Delta t) = \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) + \frac{1}{\tau^\star} \Delta t \left(\tilde{\bar{f}}_\alpha^{eq}(\cdot, t + \Delta t) - \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) \right)$

Effective viscosity: $\nu^\star = \nu + \nu_t = \frac{1}{3} \left(\tau^\star - \frac{1}{2} \right) c \Delta x \quad \text{with} \quad \tau^\star = \tau + \tau_t$

Use Smagorinsky model to evaluate $\nu_t = (C_{sm} \Delta x)^2 \bar{S}$. Filtered strain rate tensor $\bar{\mathbf{S}}_{ij} = (\partial_j \bar{u}_i + \partial_i \bar{u}_j)/2$ can also be computed as a second moment [Yu, 2004]

$$\bar{S} = \sqrt{2 \sum_{i,j} \bar{\mathbf{S}}_{ij} \bar{\mathbf{S}}_{ij}}, \qquad \bar{\mathbf{S}}_{ij} = -\frac{1}{2 \rho_0 c_s^2 \tau} \sum_a \mathbf{e}_{\alpha i} \mathbf{e}_{\alpha j} (\bar{f}_\alpha - \bar{f}_\alpha^{eq})$$

High Reynolds numbers

Pursue a large-eddy simulation approach with $\bar{f}_\alpha$ and $\bar{f}^{eq}_\alpha$, i.e.

1.) $\tilde{\bar{f}}_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = \bar{f}_\alpha(\mathbf{x}, t)$

2.) $\bar{f}_\alpha(\cdot, t + \Delta t) = \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) + \frac{1}{\tau^\star} \Delta t \left(\tilde{\bar{f}}^{eq}_\alpha(\cdot, t + \Delta t) - \tilde{\bar{f}}_\alpha(\cdot, t + \Delta t) \right)$

Effective viscosity: $\nu^\star = \nu + \nu_t = \frac{1}{3} \left(\tau^\star - \frac{1}{2} \right) c \Delta x \quad \text{with} \quad \tau^\star = \tau + \tau_t$

Use Smagorinsky model to evaluate $\nu_t = (C_{sm} \Delta x)^2 \bar{S}$. Filtered strain rate tensor $\bar{\mathbf{S}}_{ij} = (\partial_j \bar{u}_i + \partial_i \bar{u}_j)/2$ can also be computed as a second moment [Yu, 2004]

$$\bar{S} = \sqrt{2 \sum_{i,j} \bar{\mathbf{S}}_{ij} \bar{\mathbf{S}}_{ij}}, \qquad \bar{\mathbf{S}}_{ij} = -\frac{1}{2\rho_0 c_s^2 \tau} \sum_a \mathbf{e}_{\alpha i} \mathbf{e}_{\alpha j} (\bar{f}_\alpha - \bar{f}^{eq}_\alpha)$$

Special note: LBM is normally implemented on the unit lattice with $\Delta \tilde{x} = \Delta \tilde{t} = 1$ and normalization

$$\frac{\Delta x}{l_0} = 1, \quad \frac{\Delta t}{t_0} = 1 \longrightarrow c = 1$$

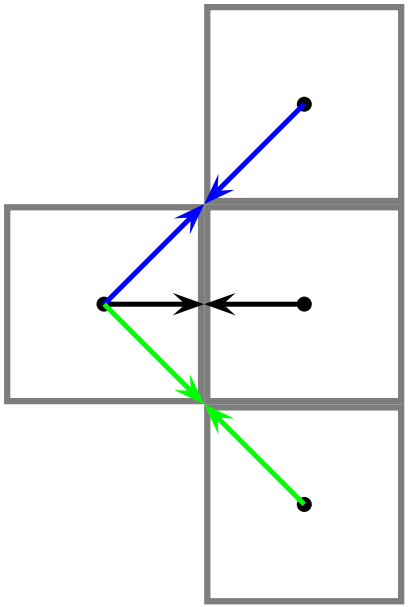
giving a lattice viscosity $\tilde{\nu} = \frac{1}{3} \left(\tau - \frac{1}{2} \right)$ and lattice sound speed $\tilde{c}_s = \frac{1}{\sqrt{3}}$.
This requires special attention when implementing turbulence models, etc.

Initial and boundary conditions

- Initial conditions are constructed as $f_\alpha^{eq}(\rho(t=0), \mathbf{u}(t=0))$

Simple boundary conditions:

No-slip

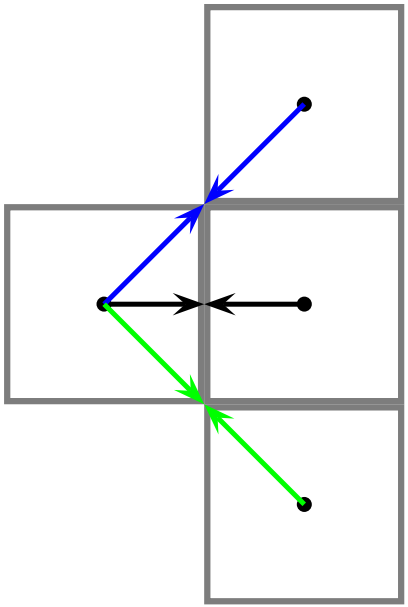


Initial and boundary conditions

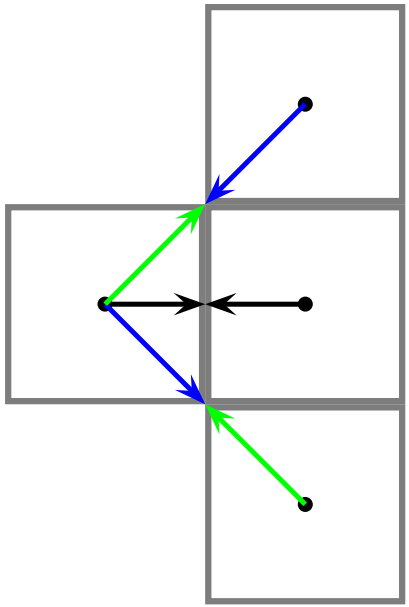
- Initial conditions are constructed as $f_\alpha^{eq}(\rho(t=0), \mathbf{u}(t=0))$

Simple boundary conditions:

No-slip



Slip

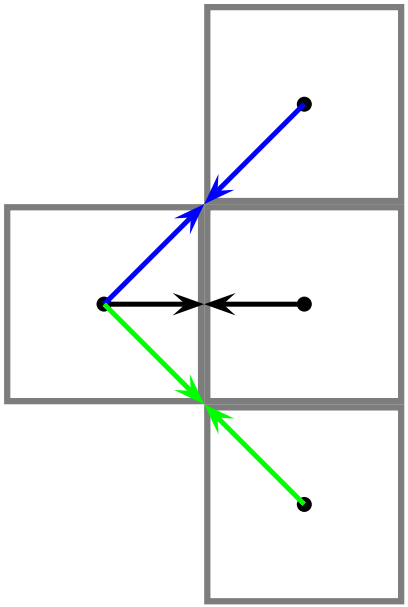


Initial and boundary conditions

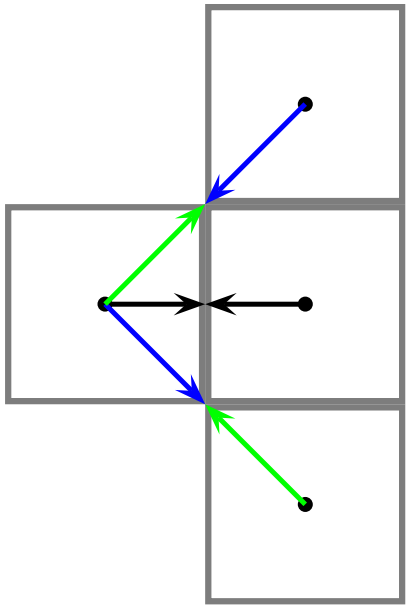
- Initial conditions are constructed as $f_\alpha^{eq}(\rho(t=0), \mathbf{u}(t=0))$

Simple boundary conditions:

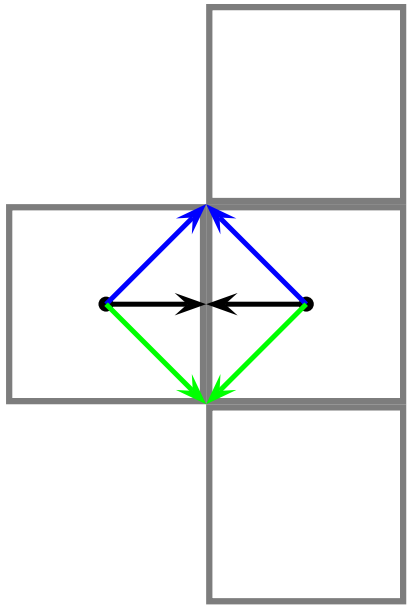
No-slip



Slip



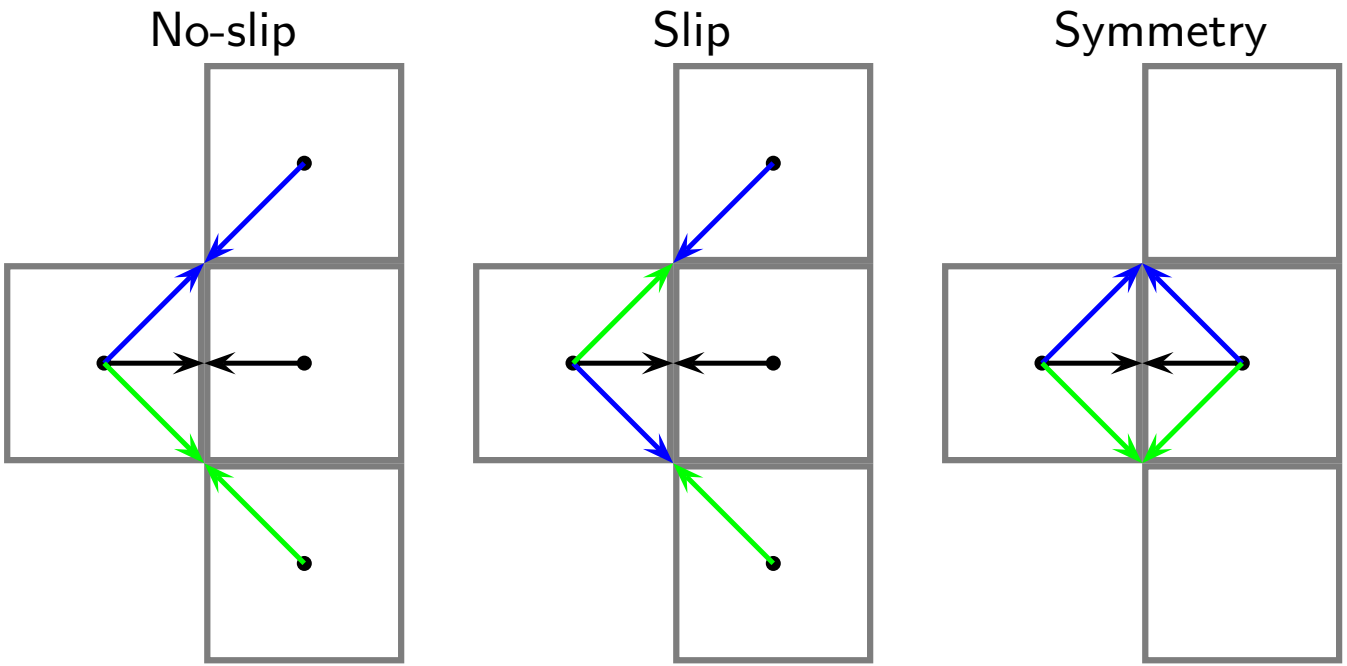
Symmetry



Initial and boundary conditions

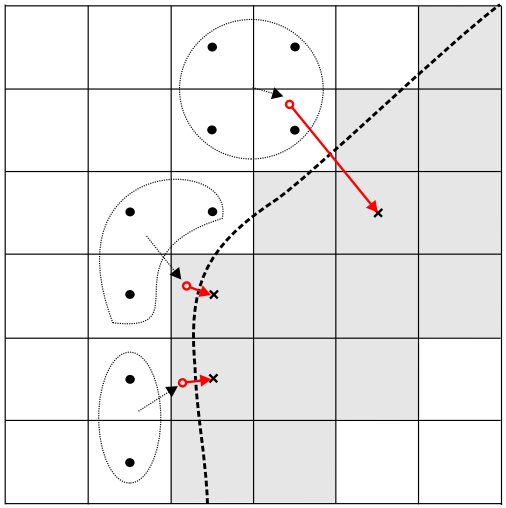
- Initial conditions are constructed as $f_\alpha^{eq}(\rho(t=0), \mathbf{u}(t=0))$

Simple boundary conditions:



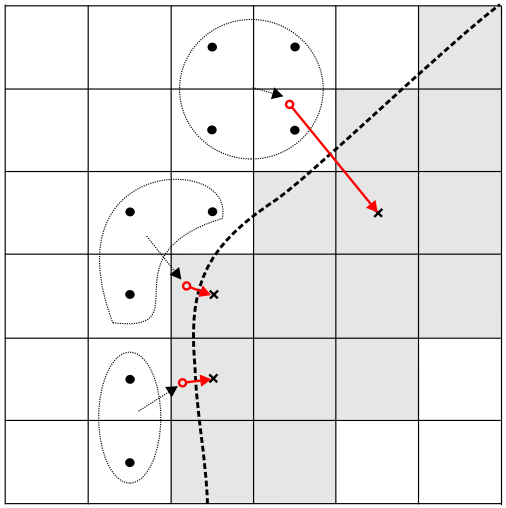
- Outlet basically copies all distributions (zero gradient)
- Inlet and pressure boundary conditions use f_α^{eq}

Level-set method for boundary embedding



- Implicit boundary representation via distance function φ, normal $\mathbf{n} = \nabla\varphi/|\nabla\varphi|$.
- Complex boundary moving with local velocity $\mathbf{w}$, treat interface as moving rigid wall.
- Construction of macro-values in embedded boundary cells by interpolation / extrapolation.

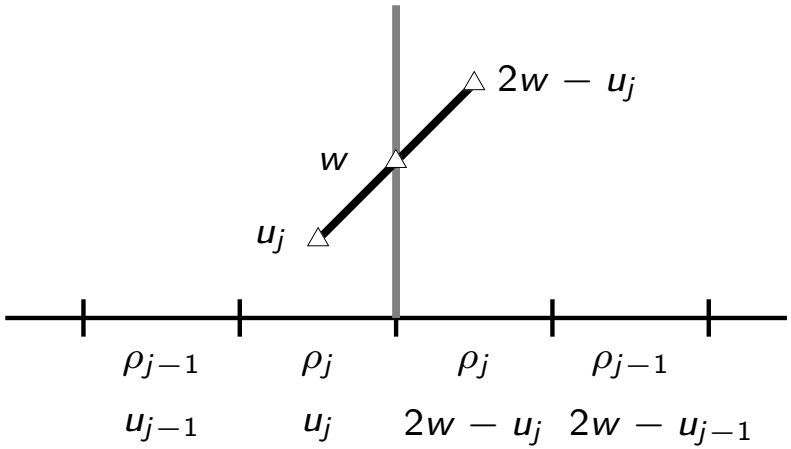
Level-set method for boundary embedding



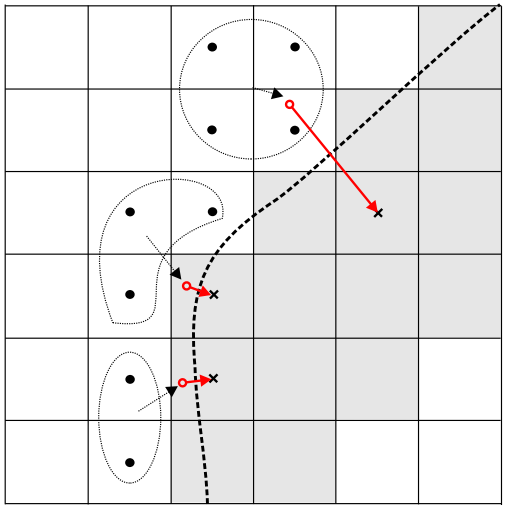
- Implicit boundary representation via distance function φ, normal $\mathbf{n} = \nabla\varphi/|\nabla\varphi|$.
- Complex boundary moving with local velocity $\mathbf{w}$, treat interface as moving rigid wall.
- Construction of macro-values in embedded boundary cells by interpolation / extrapolation.

Interpolate / constant value extrapolate values at

$$\tilde{\mathbf{x}} = \mathbf{x} + 2\varphi\mathbf{n}$$



Level-set method for boundary embedding



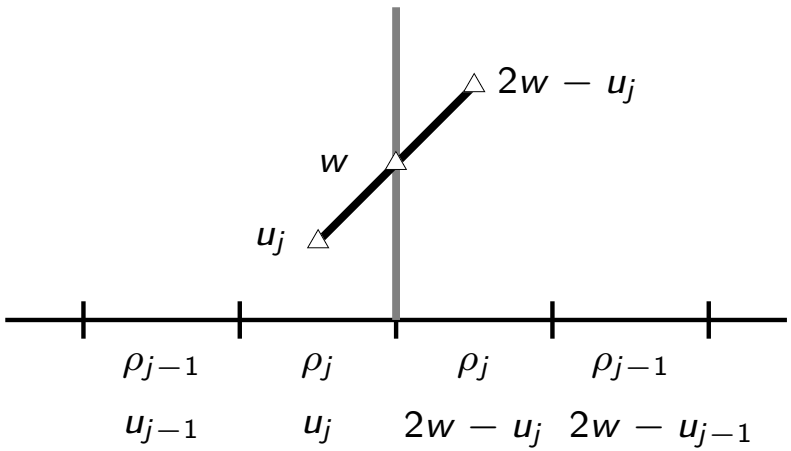
- Implicit boundary representation via distance function φ, normal $\mathbf{n} = \nabla\varphi/|\nabla\varphi|$.
- Complex boundary moving with local velocity $\mathbf{w}$, treat interface as moving rigid wall.
- Construction of macro-values in embedded boundary cells by interpolation / extrapolation.

Interpolate / constant value extrapolate values at

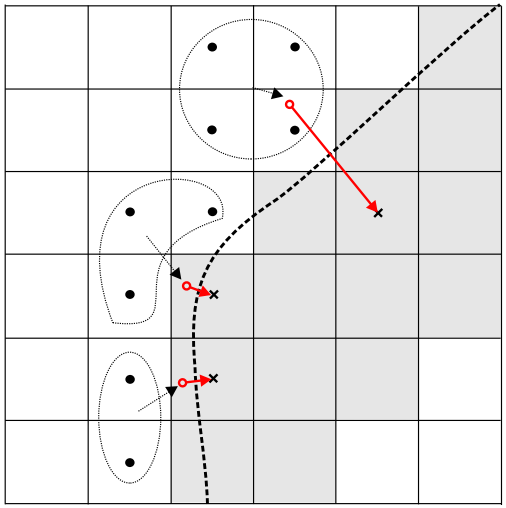
$$\tilde{\mathbf{x}} = \mathbf{x} + 2\varphi\mathbf{n}$$

Macro-velocity in ghost cells:
No-slip: $\mathbf{u}' = 2\mathbf{w} - \mathbf{u}$
Slip:

$$\begin{aligned}\mathbf{u}' &= (2\mathbf{w}\cdot\mathbf{n} - \mathbf{u}\cdot\mathbf{n})\mathbf{n} + (\mathbf{u}\cdot\mathbf{t})\mathbf{t} \\ &= 2\left((\mathbf{w}-\mathbf{u})\cdot\mathbf{n}\right)\mathbf{n} + \mathbf{u}\end{aligned}$$



Level-set method for boundary embedding



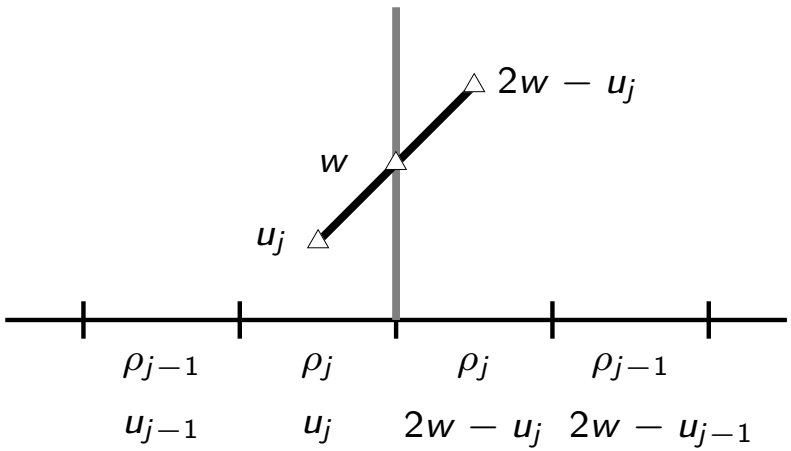
- Implicit boundary representation via distance function φ, normal $\mathbf{n} = \nabla\varphi/|\nabla\varphi|$.
- Complex boundary moving with local velocity $\mathbf{w}$, treat interface as moving rigid wall.
- Construction of macro-values in embedded boundary cells by interpolation / extrapolation.
- Then use $f_\alpha^{eq}(\rho', \mathbf{u}')$ to construct distributions in embedded ghost cells.
- 2nd order improvements possible, cf. [Peng and Luo, 2008].

Interpolate / constant value extrapolate values at

$$\tilde{\mathbf{x}} = \mathbf{x} + 2\varphi\mathbf{n}$$

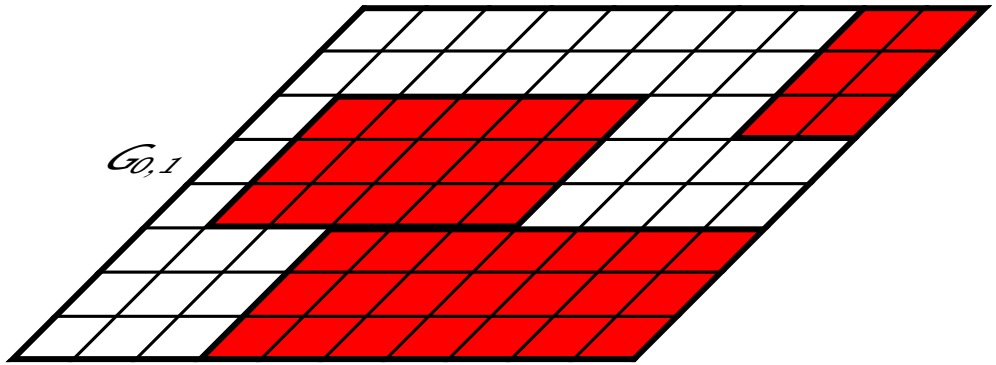
Macro-velocity in ghost cells:
No-slip: $\mathbf{u}' = 2\mathbf{w} - \mathbf{u}$
Slip:

$$\begin{aligned}\mathbf{u}' &= (2\mathbf{w}\cdot\mathbf{n} - \mathbf{u}\cdot\mathbf{n})\mathbf{n} + (\mathbf{u}\cdot\mathbf{t})\mathbf{t} \\ &= 2\left((\mathbf{w}-\mathbf{u})\cdot\mathbf{n}\right)\mathbf{n} + \mathbf{u}\end{aligned}$$



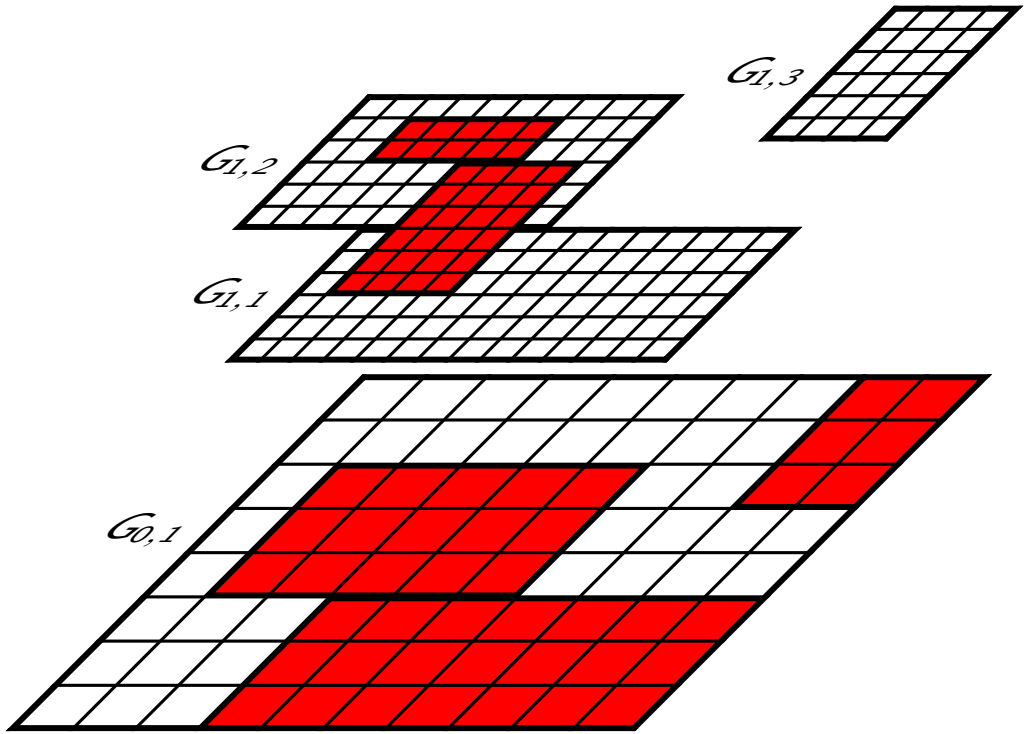
Block-structured adaptive mesh refinement (SAMR)

- Refined blocks overlay coarser ones



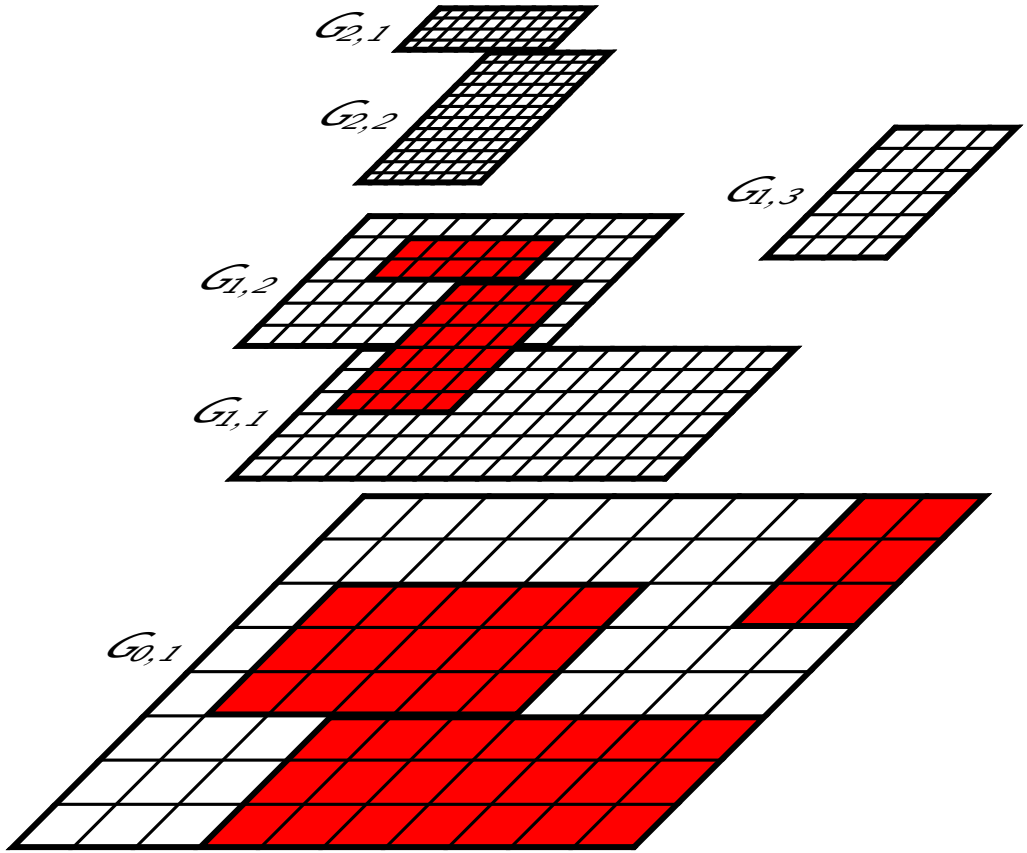
Block-structured adaptive mesh refinement (SAMR)

- Refined blocks overlay coarser ones



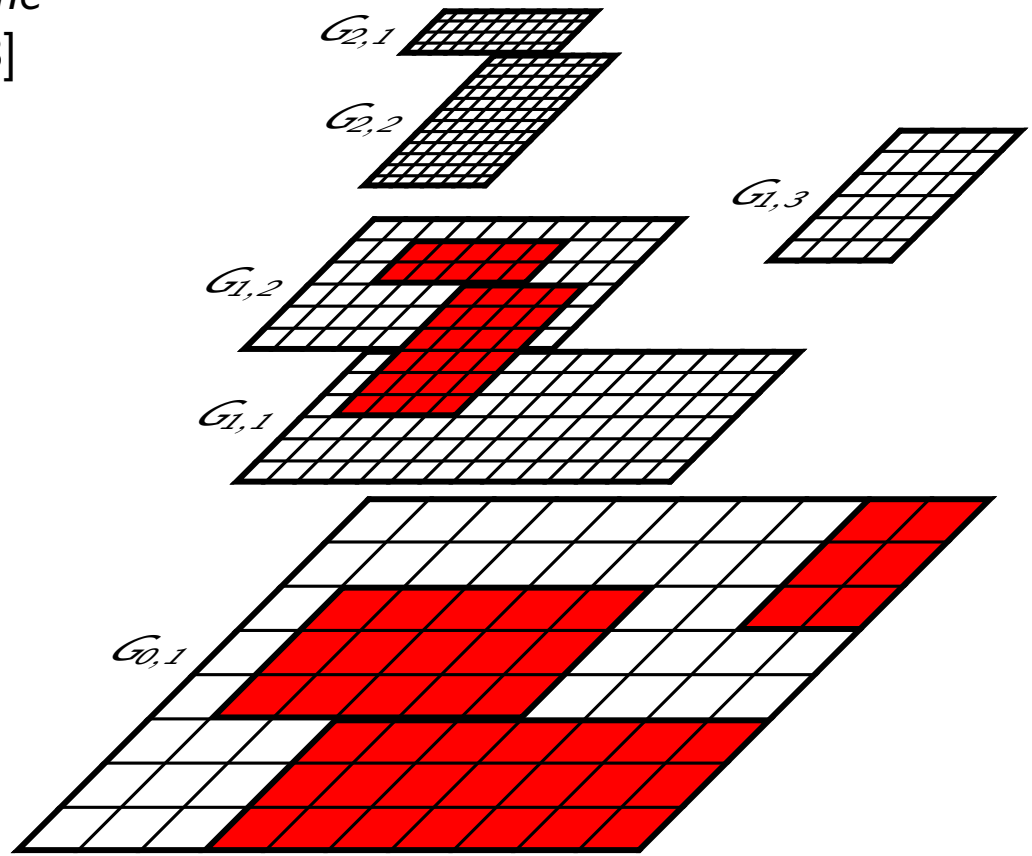
Block-structured adaptive mesh refinement (SAMR)

- Refined blocks overlay coarser ones



Block-structured adaptive mesh refinement (SAMR)

- Refined blocks overlay coarser ones
- Recursive refinement in space *and time* by factor r_l [Berger and Colella, 1988] ideal for LBM

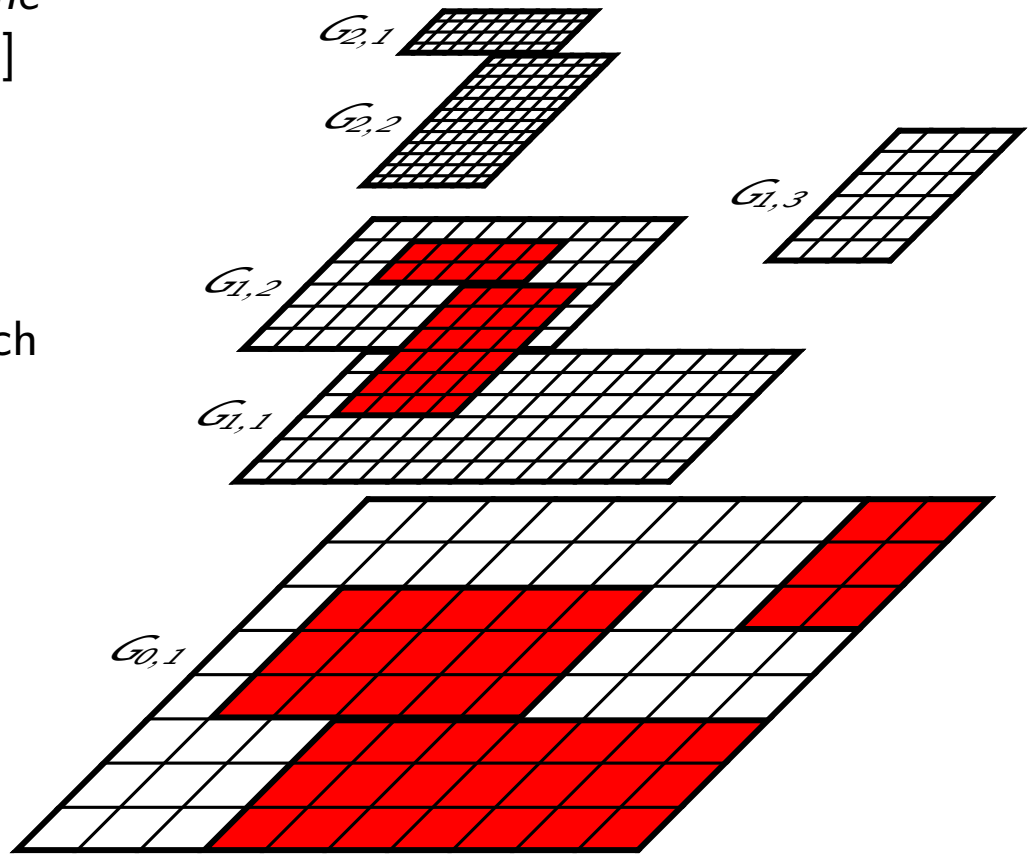


Block-structured adaptive mesh refinement (SAMR)

- Refined blocks overlay coarser ones
- Recursive refinement in space *and time* by factor r_l [Berger and Colella, 1988] ideal for LBM
- Block (aka patch) based data structures

\+ Numerical scheme only for single patch necessary

\+ Most efficient LBM implementation with patch-wise for-loops



Block-structured adaptive mesh refinement (SAMR)

- Refined blocks overlay coarser ones
- Recursive refinement in space *and time* by factor r_l [Berger and Colella, 1988] ideal for LBM
- Block (aka patch) based data structures

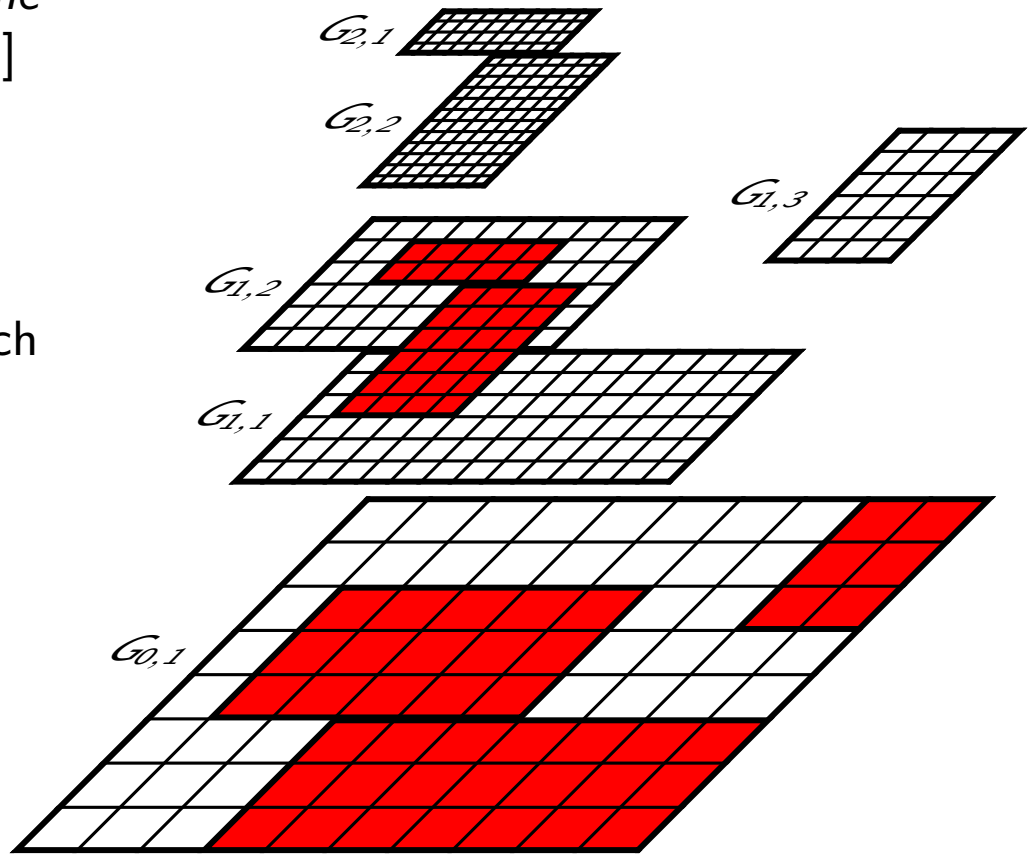
\+ Numerical scheme only for single patch necessary

\+ Most efficient LBM implementation with patch-wise for-loops

\+ Cache efficient

- Spatial interpolation and averaging can be used unaltered

\- Cluster-algorithm necessary



AMROC software

- Adaptive mesh refinement in object-oriented C++ – tailored for explicit block-based finite volume methods
- Many shock-capturing methods (MUSCL, (hybrid) WENO, etc.) implemented for complex flux functions.

AMROC software

- Adaptive mesh refinement in object-oriented C++ – tailored for explicit block-based finite volume methods
- Many shock-capturing methods (MUSCL, (hybrid) WENO, etc.) implemented for complex flux functions.
- Drives Virtual Test Facility (VTF) FSI software.
- Targets strongly driven problems (shocks, blast, detonations)
- Geometry embedding via ghost fluid techniques and level set functions. Distance computation with CPT algorithm [Mauch, 2000].

AMROC software

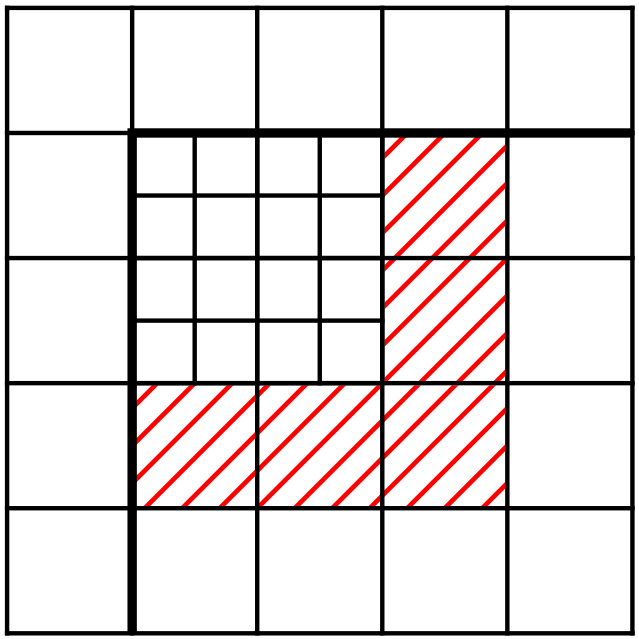
- Adaptive mesh refinement in object-oriented C++ – tailored for explicit block-based finite volume methods
- Many shock-capturing methods (MUSCL, (hybrid) WENO, etc.) implemented for complex flux functions.
- Drives Virtual Test Facility (VTF) FSI software.
- Targets strongly driven problems (shocks, blast, detonations)
- Geometry embedding via ghost fluid techniques and level set functions. Distance computation with CPT algorithm [Mauch, 2000].
- $\sim 430,000$ LOC in C++, C, Fortran-77, Fortran-90.
- Version V2.0 at http://www.cacr.caltech.edu/asc. V1.1 (no complex boundaries) still at http://amroc.sourceforge.net.
- Version used here V3.0 with significantly enhanced parallelization (V2.1 not released).
- Papers: [Deiterding, 2011, Deiterding and Wood, 2013, Deiterding et al., 2009, Deiterding et al., 2007, Deiterding et al., 2006]

AMROC software

- Adaptive mesh refinement in object-oriented C++ – tailored for explicit block-based finite volume methods
- Many shock-capturing methods (MUSCL, (hybrid) WENO, etc.) implemented for complex flux functions.
- Drives Virtual Test Facility (VTF) FSI software.
- Targets strongly driven problems (shocks, blast, detonations)
- Geometry embedding via ghost fluid techniques and level set functions. Distance computation with CPT algorithm [Mauch, 2000].
- $\sim 430,000$ LOC in C++, C, Fortran-77, Fortran-90.
- Version V2.0 at http://www.cacr.caltech.edu/asc. V1.1 (no complex boundaries) still at http://amroc.sourceforge.net.
- Version used here V3.0 with significantly enhanced parallelization (V2.1 not released).
- Papers: [Deiterding, 2011, Deiterding and Wood, 2013, Deiterding et al., 2009, Deiterding et al., 2007, Deiterding et al., 2006]
- Lattice Boltzmann update is implemented like any other finite volume method but without returning numerical fluxes.

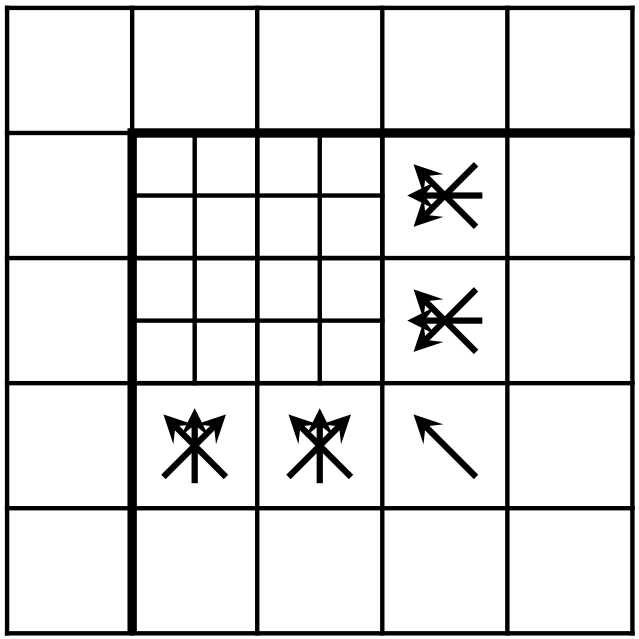
Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$



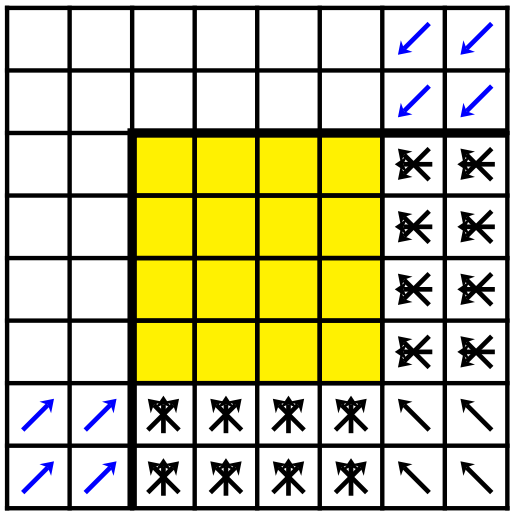
Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.



Adaptive LBM

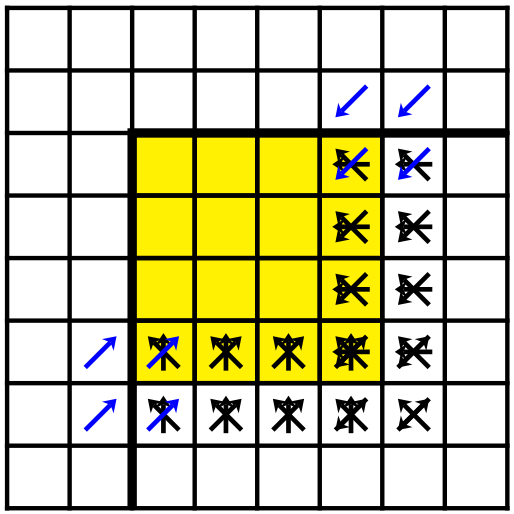
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.



$f_{\alpha,in}^{f,n}$

Adaptive LBM

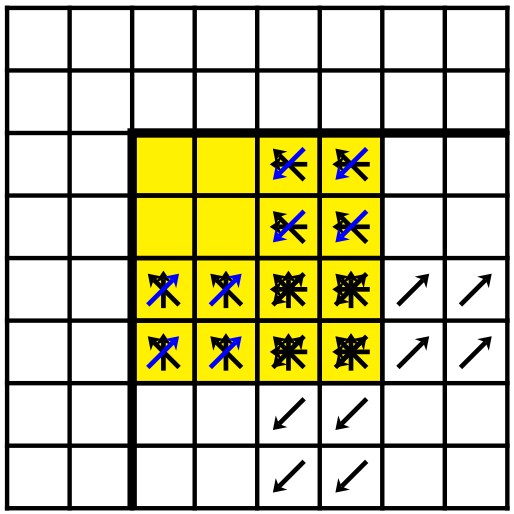
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.



$\tilde{f}_{\alpha,in}^{f,n}$

Adaptive LBM

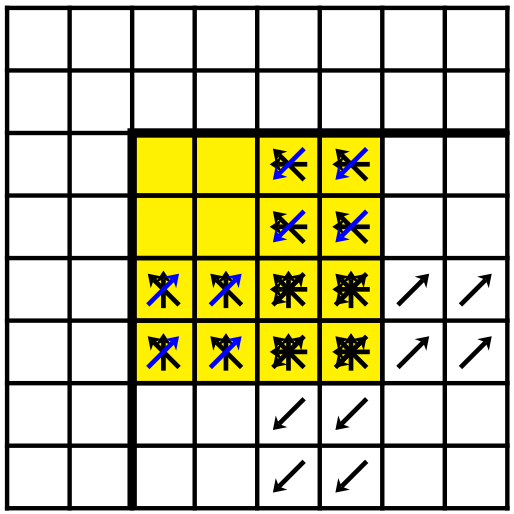
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



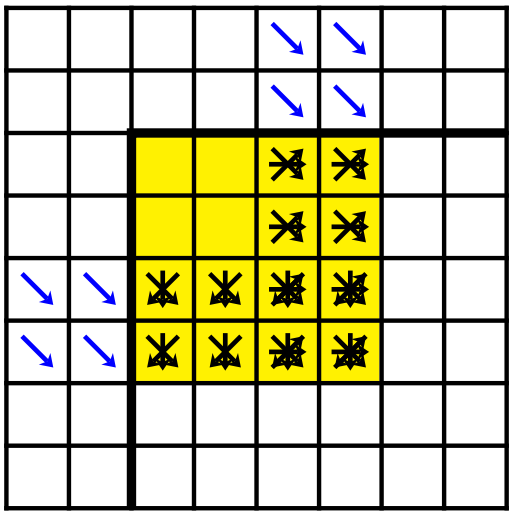
$\tilde{f}_{\alpha,in}^{f,n+1/2}$

Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



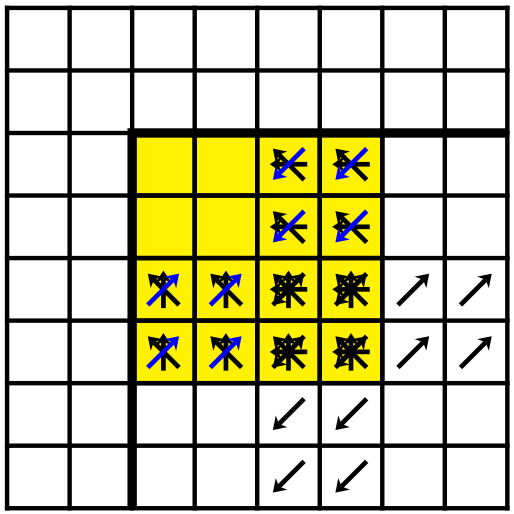
$\tilde{f}_{\alpha,in}^{f,n+1/2}$



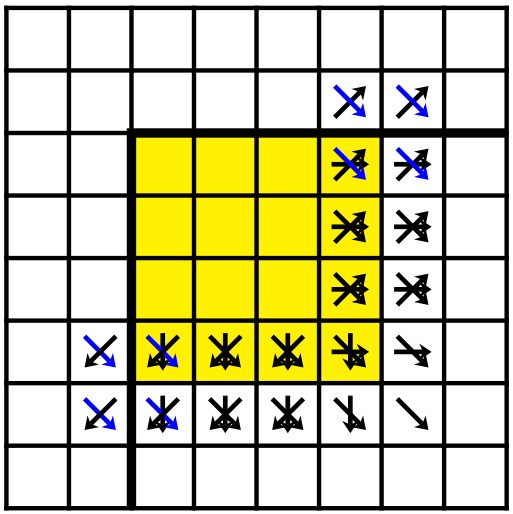
$f_{\alpha,out}^{f,n}$

Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



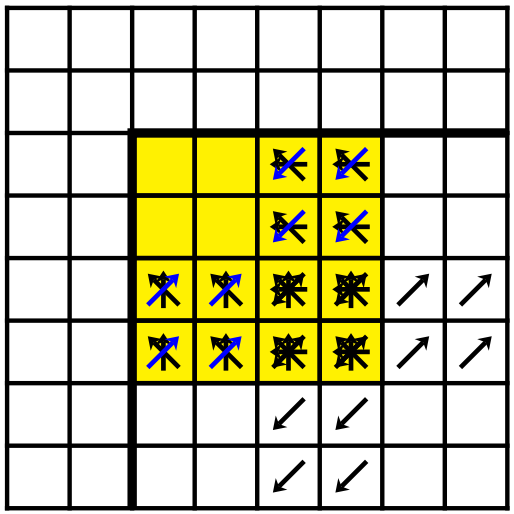
$\tilde{f}_{\alpha,in}^{f,n+1/2}$



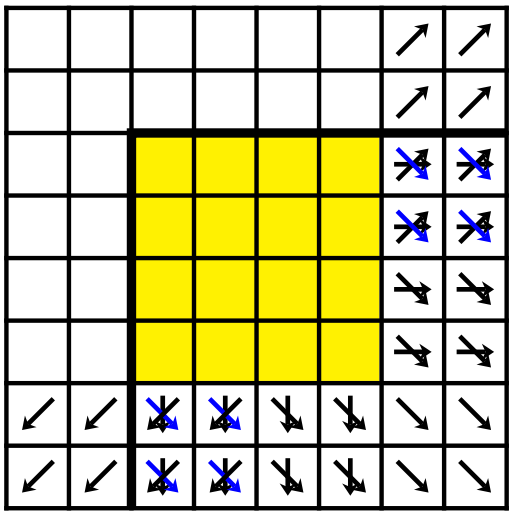
$\tilde{f}_{\alpha,out}^{f,n}$

Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



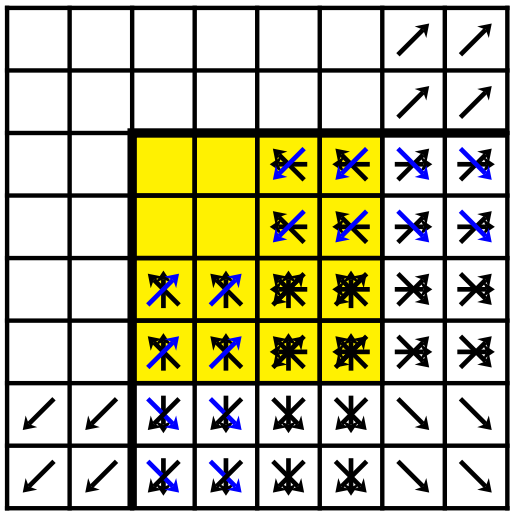
$\tilde{f}_{\alpha,in}^{f,n+1/2}$



$\tilde{f}_{\alpha,out}^{f,n+1/2}$

Adaptive LBM

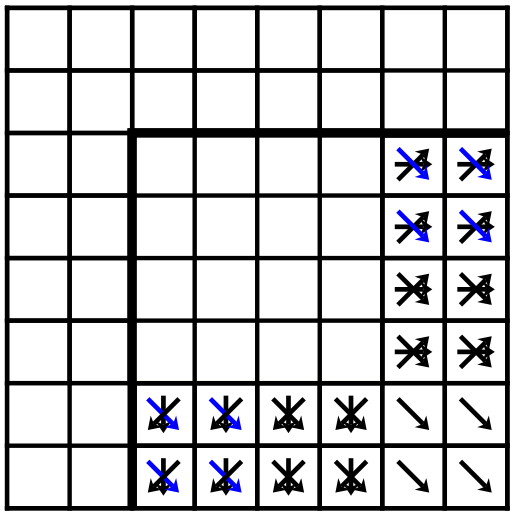
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



$\tilde{f}_{\alpha,out}^{f,n+1/2}, \tilde{f}_{\alpha,in}^{f,n+1/2}$

Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.

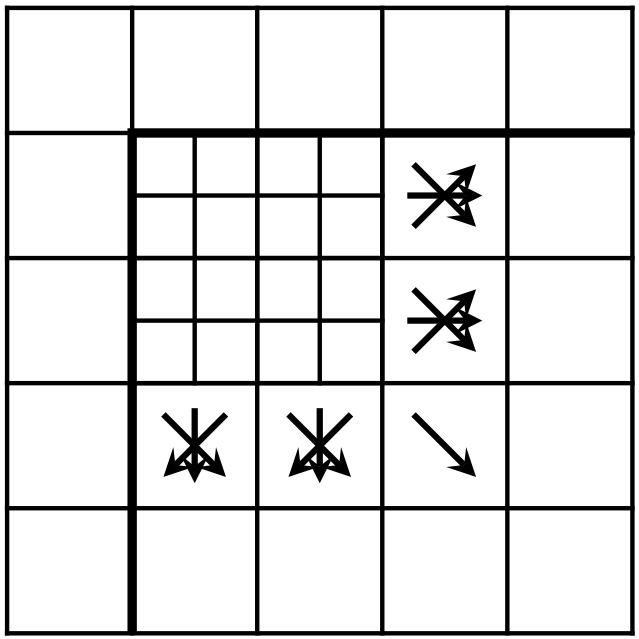


$\tilde{f}_{\alpha,out}^{f,n+1/2}$, $\tilde{f}_{\alpha,out}^{f,n}$

5. Average $\tilde{f}_{\alpha,out}^{f,n+1/2}$ (inner halo layer), $\tilde{f}_{\alpha,out}^{f,n}$ (outer halo layer) to obtain $\tilde{f}_{\alpha,out}^{C,n}$.

Adaptive LBM

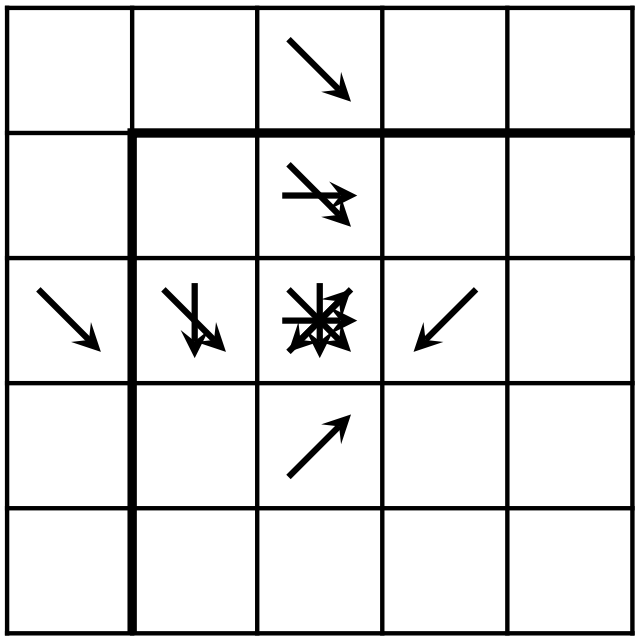
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



5. Average $\tilde{f}_{\alpha,out}^{f,n+1/2}$ (inner halo layer), $\tilde{f}_{\alpha,out}^{f,n}$ (outer halo layer) to obtain $\tilde{f}_{\alpha,out}^{C,n}$.

Adaptive LBM

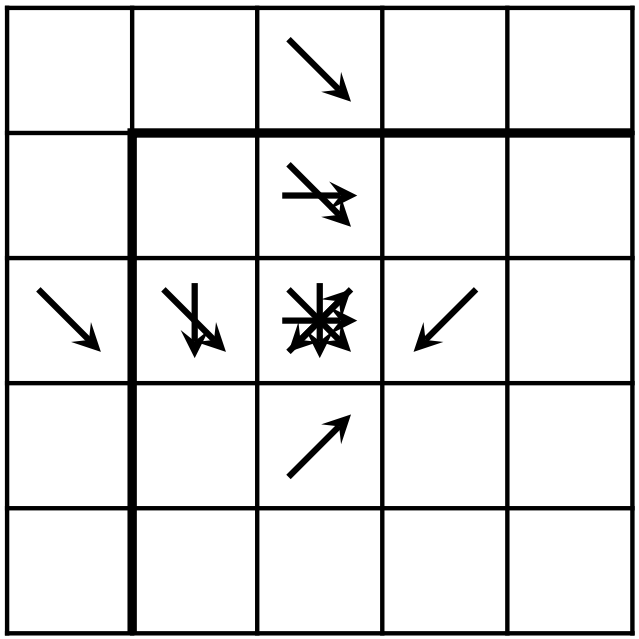
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



5. Average $\tilde{f}_{\alpha,out}^{f,n+1/2}$ (inner halo layer), $\tilde{f}_{\alpha,out}^{f,n}$ (outer halo layer) to obtain $\tilde{f}_{\alpha,out}^{C,n}$.
6. Revert transport into halos: $\bar{f}_{\alpha,out}^{C,n} := \mathcal{T}^{-1}(\tilde{f}_{\alpha,out}^{C,n})$

Adaptive LBM

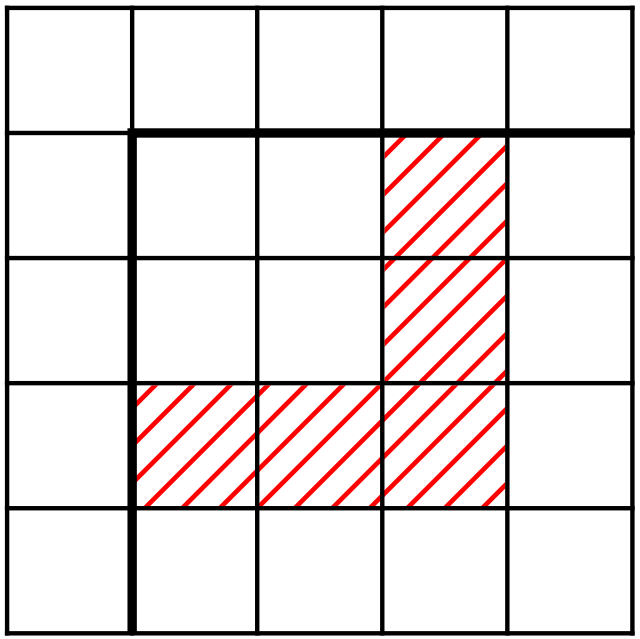
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



5. Average $\tilde{f}_{\alpha,out}^{f,n+1/2}$ (inner halo layer), $\tilde{f}_{\alpha,out}^{f,n}$ (outer halo layer) to obtain $\tilde{f}_{\alpha,out}^{C,n}$.
6. Revert transport into halos: $\bar{f}_{\alpha,out}^{C,n} := \mathcal{T}^{-1}(\tilde{f}_{\alpha,out}^{C,n})$
7. Parallel synchronization of $f_\alpha^{C,n}$, $\bar{f}_{\alpha,out}^{C,n}$

Adaptive LBM

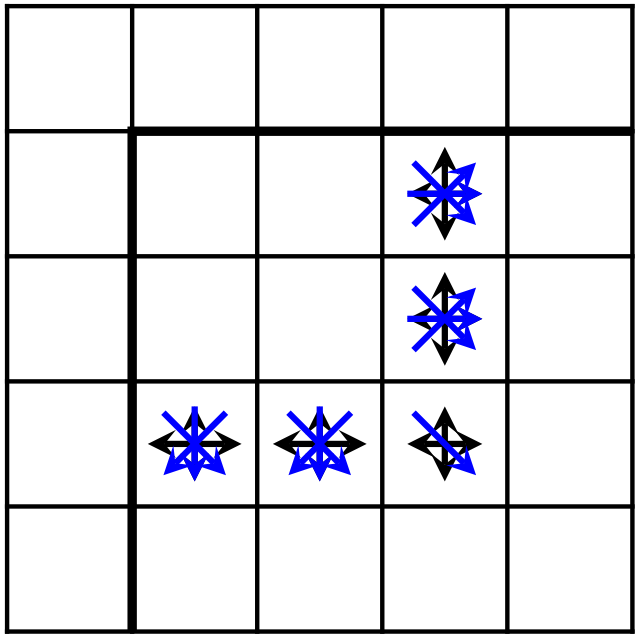
1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.



5. Average $\tilde{f}_{\alpha,out}^{f,n+1/2}$ (inner halo layer), $\tilde{f}_{\alpha,out}^{f,n}$ (outer halo layer) to obtain $\tilde{f}_{\alpha,out}^{C,n}$.
6. Revert transport into halos: $\bar{f}_{\alpha,out}^{C,n} := \mathcal{T}^{-1}(\tilde{f}_{\alpha,out}^{C,n})$
7. Parallel synchronization of $f_\alpha^{C,n}$, $\bar{f}_{\alpha,out}^{C,n}$
8. Cell-wise update where correction is needed: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n}, \bar{f}_{\alpha,out}^{C,n})$

Adaptive LBM

1. Complete update on coarse grid: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n})$
2. Interpolate $f_{\alpha,in}^{C,n}$ onto $f_{\alpha,in}^{f,n}$ to fill fine halos. Set physical boundary conditions.
3. $\tilde{f}_\alpha^{f,n} := \mathcal{T}(f_\alpha^{f,n})$ on whole fine mesh. $f_\alpha^{f,n+1/2} := \mathcal{C}(\tilde{f}_\alpha^{f,n})$ in interior.
4. $\tilde{f}_\alpha^{f,n+1/2} := \mathcal{T}(f_\alpha^{f,n+1/2})$ on whole fine mesh. $f_\alpha^{f,n+1} := \mathcal{C}(\tilde{f}_\alpha^{f,n+1/2})$ in interior.

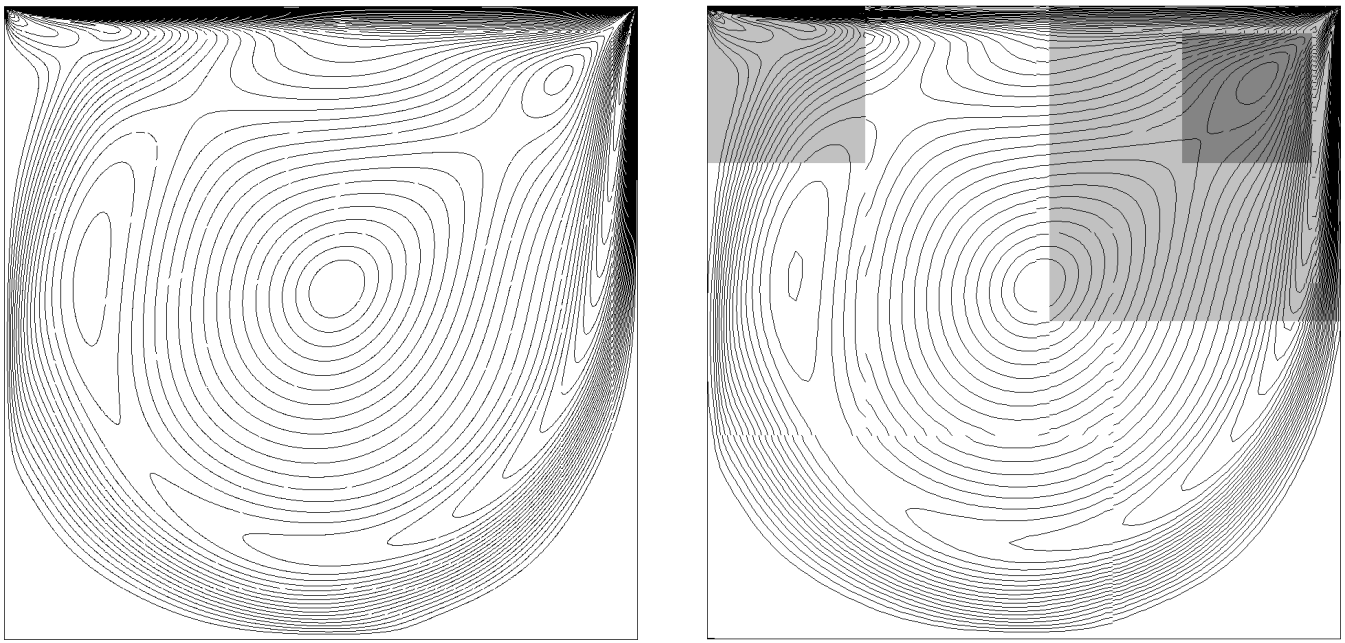


5. Average $\tilde{f}_{\alpha,out}^{f,n+1/2}$ (inner halo layer), $\tilde{f}_{\alpha,out}^{f,n}$ (outer halo layer) to obtain $\tilde{f}_{\alpha,out}^{C,n}$.
6. Revert transport into halos: $\bar{f}_{\alpha,out}^{C,n} := \mathcal{T}^{-1}(\tilde{f}_{\alpha,out}^{C,n})$
7. Parallel synchronization of $f_\alpha^{C,n}$, $\bar{f}_{\alpha,out}^{C,n}$
8. Cell-wise update where correction is needed: $f_\alpha^{C,n+1} := \mathcal{CT}(f_\alpha^{C,n}, \bar{f}_{\alpha,out}^{C,n})$

Algorithm equivalent to [Chen et al., 2006].

Driven cavity

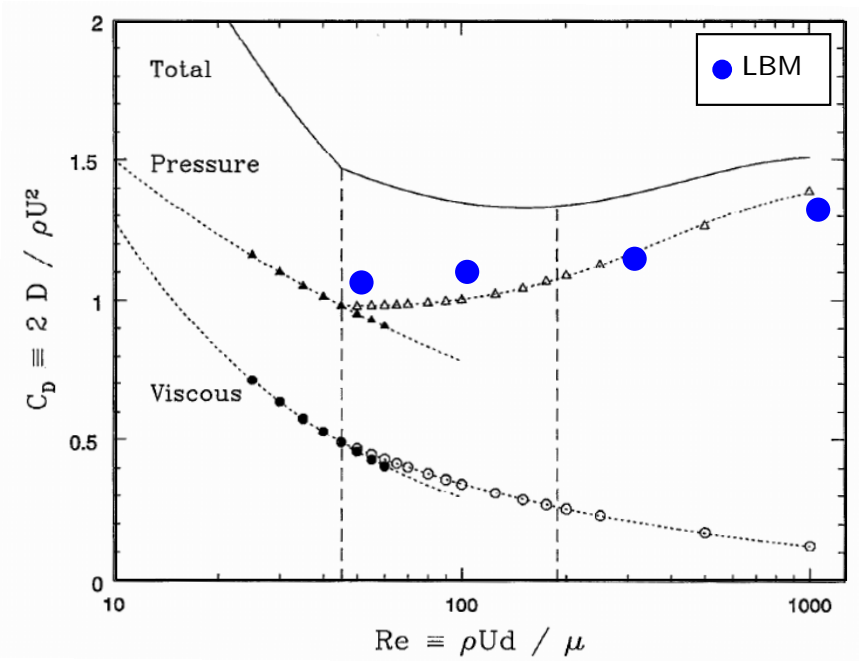
- $\mathrm{Re} = 1500$ in air, $\nu = 1.5 \cdot 10^{-5}\,\mathrm{m}^2/\mathrm{s}$, $u = 22.5\,\mathrm{m}/\mathrm{s}$.
- Domain size $1\,\mathrm{mm} \times 1\,\mathrm{mm}$.
- Reference computation uses 800×800 lattice.
- 588,898 time steps to $t_e = 5 \cdot 10^{-3}\,\mathrm{s}$, $\sim 35\,\mathrm{h}$ CPU.
- Statically adaptive computation uses 100×100 lattice with $r_{1,2} = 2$.
- 294,452 time steps to $t_e = 5 \cdot 10^{-3}\,\mathrm{s}$ on finest level.



Isolines of density. Left: reference, right on refinement at t_e.

Flow over 2D cylinder, $d = 2\,\mathrm{cm}$

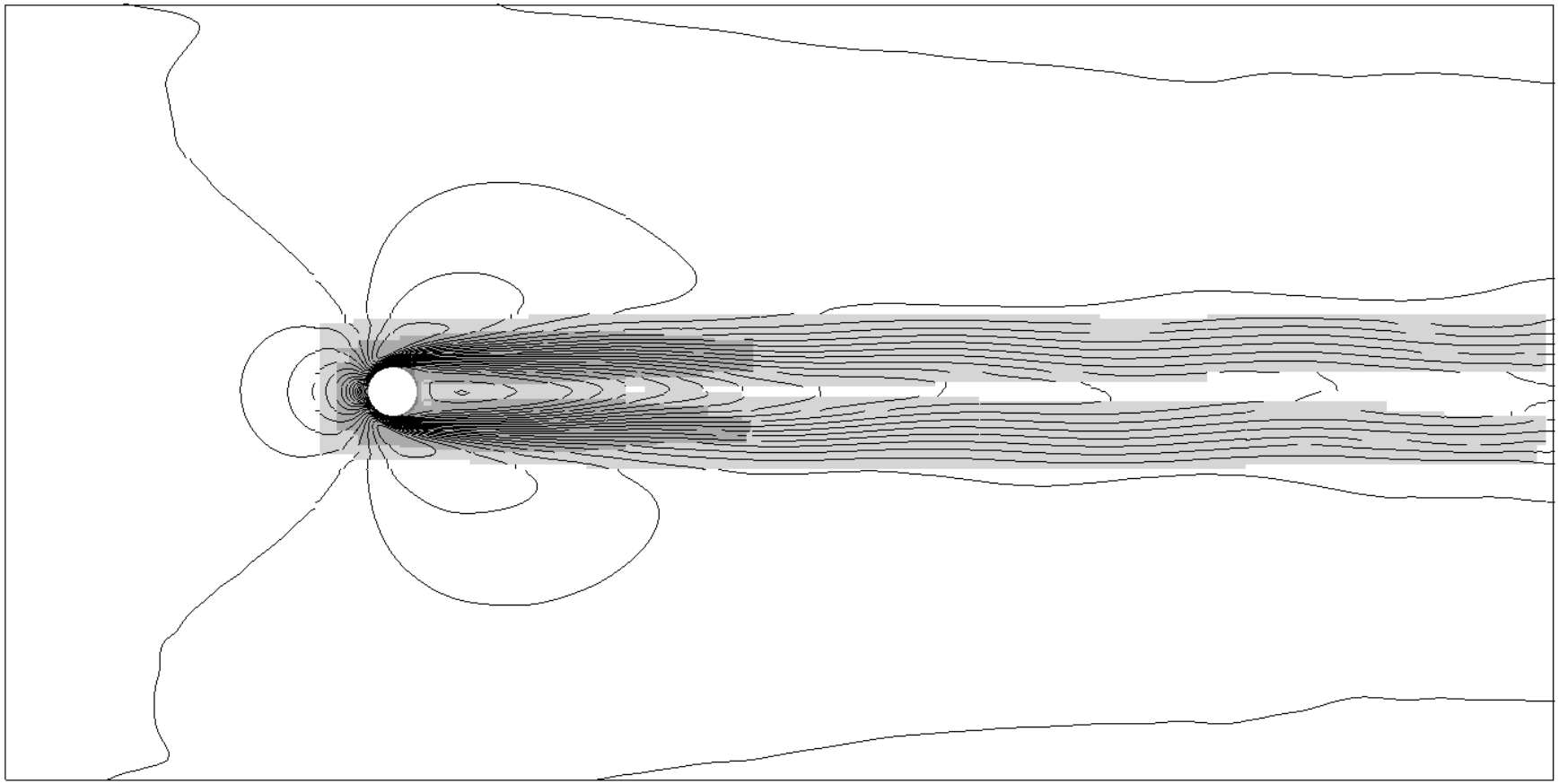
- Air with
  $\nu = 1.61 \cdot 10^{-5}\,\mathrm{m}^2/\mathrm{s}$,
  $\rho = 1.205\,\mathrm{kg}/\mathrm{m}^3$
- Domain size
  $[-8d, 24d] \times [-8d, 8d]$
- Dynamic refinement based on velocity. Last level to refine structure further.
- Inflow from left. Characteristic boundary conditions [Schlaffer, 2013] elsewhere.



- Base lattice 320×160, 3 additional levels with factors $r_l = 2, 4, 4$.
- Resolution: ~ 320 points in diameter d
- Computation of C_D on 400 equidistant points along circle and averaged over time. Comparison above with [Henderson, 1995].

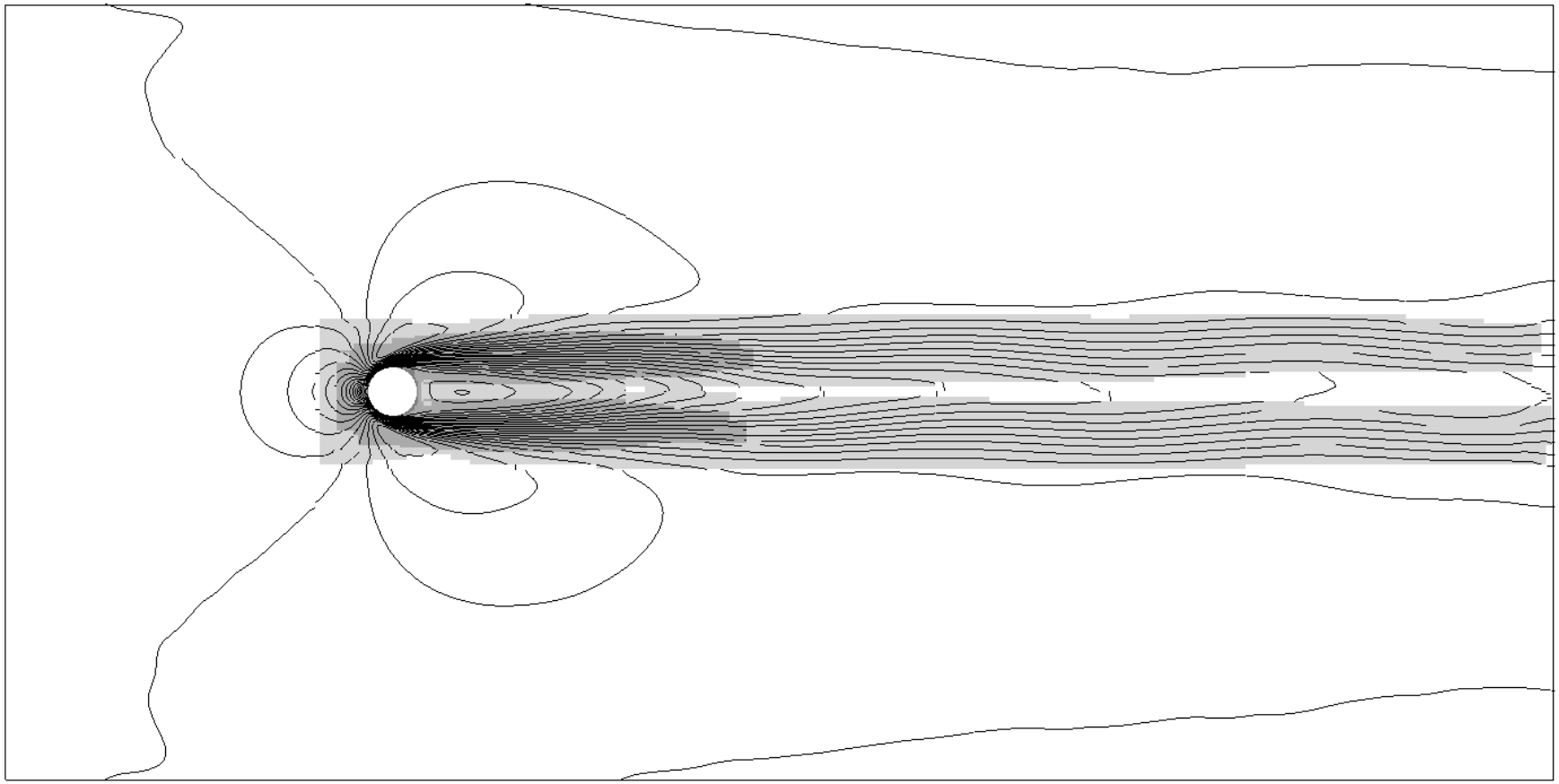
Flow over cylinder in 2d - $\mathrm{Re} = 50$, $u = 0.04025\,\mathrm{m/s}$

Isolines on refinement



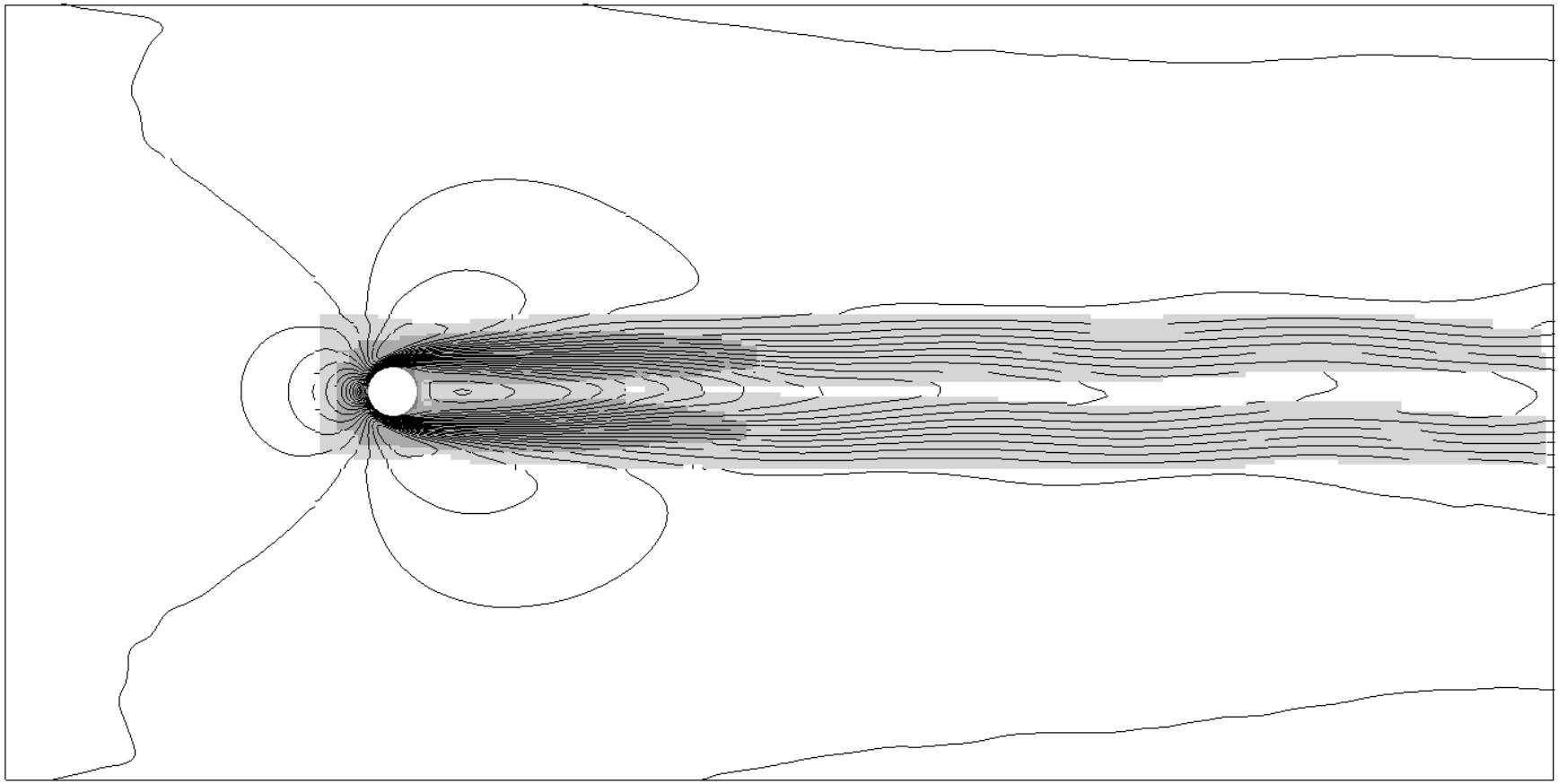
Flow over cylinder in 2d - $\mathrm{Re} = 50$, $u = 0.04025\,\mathrm{m/s}$

Isolines on refinement



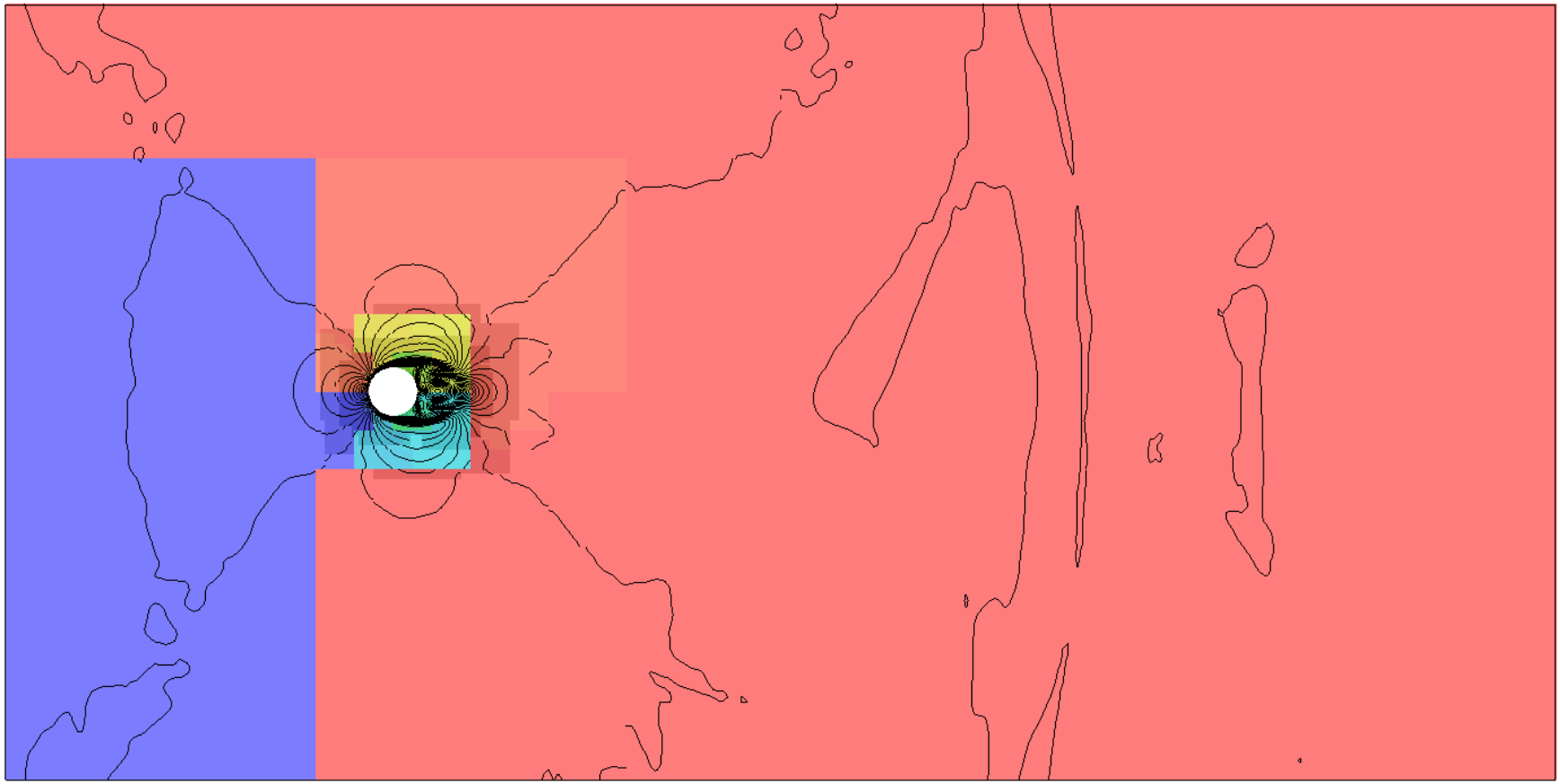
Flow over cylinder in 2d - $\mathrm{Re} = 50$, $u = 0.04025\,\mathrm{m/s}$

Isolines on refinement



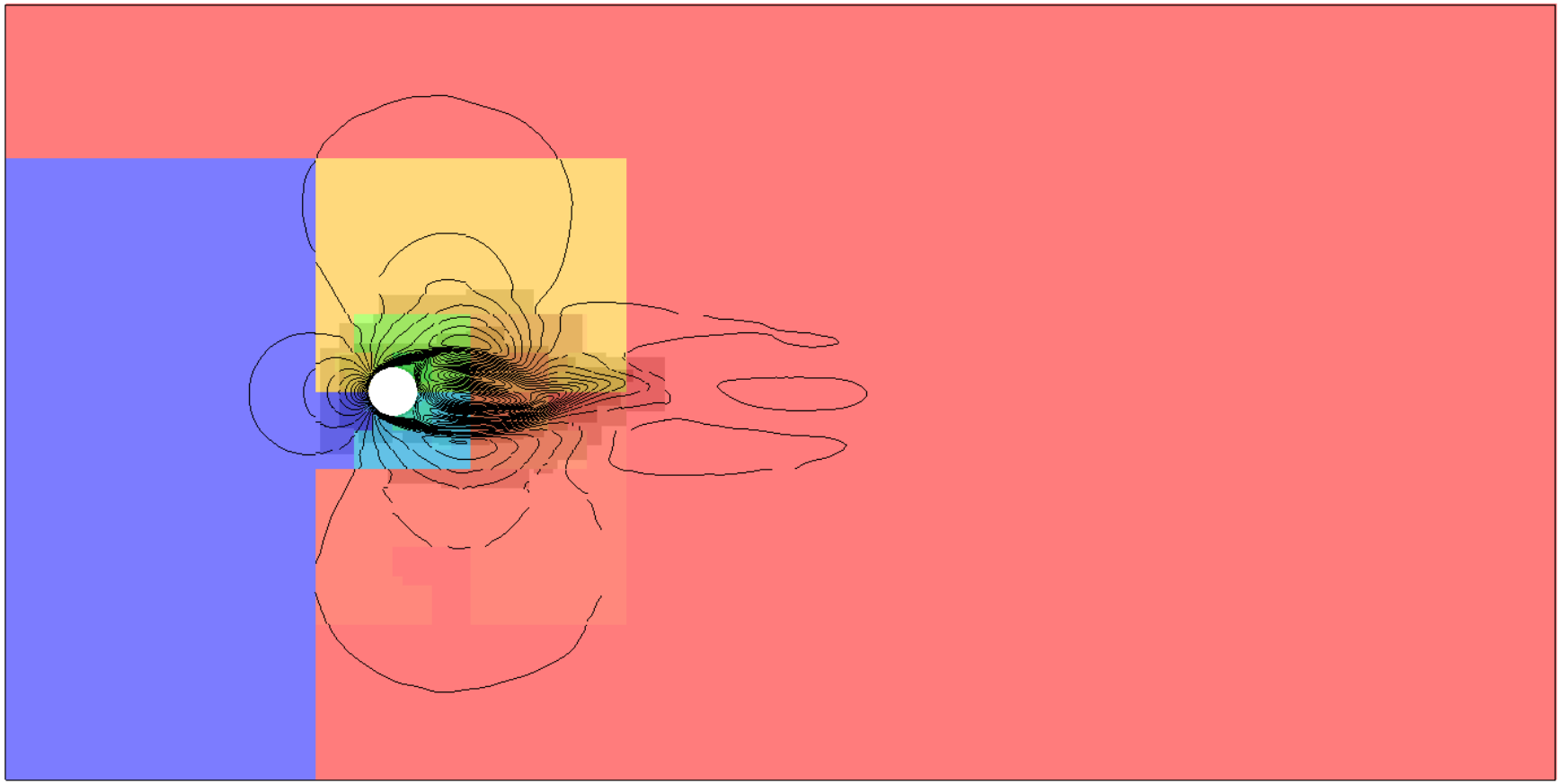
Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors



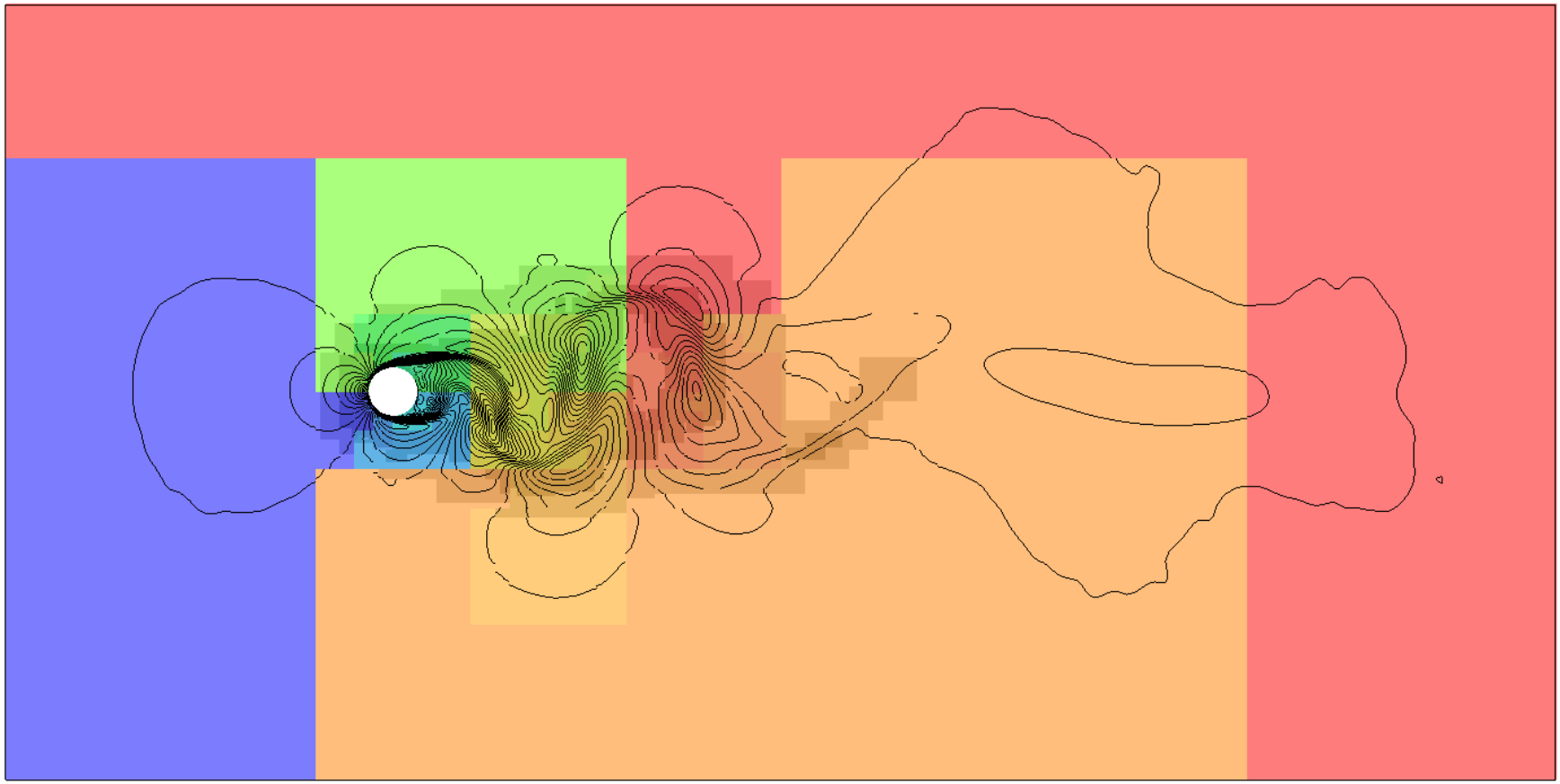
Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors



Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors



Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors

Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors

Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

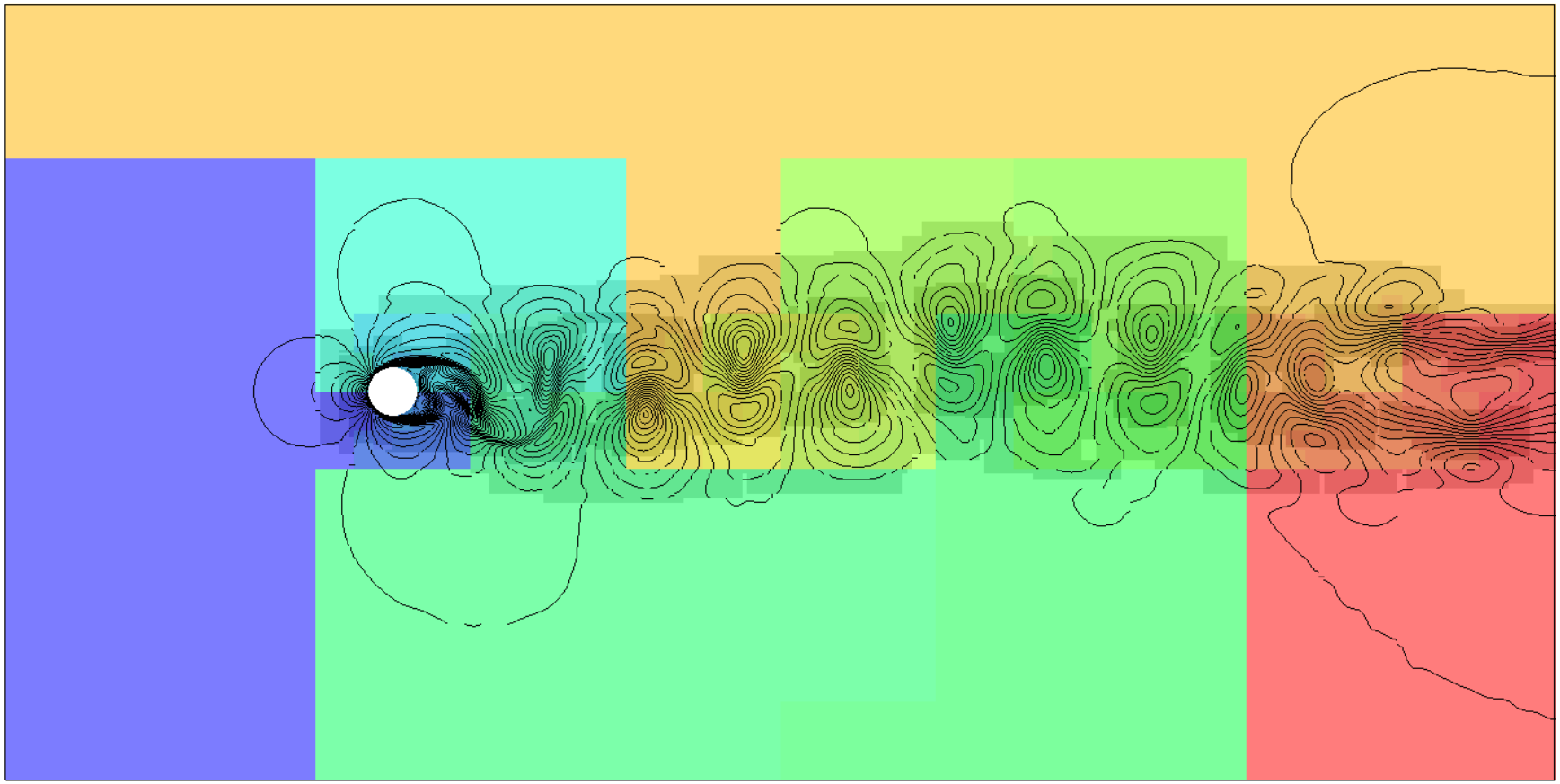
Isolines on refinement and distribution to processors

Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors

Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors



Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

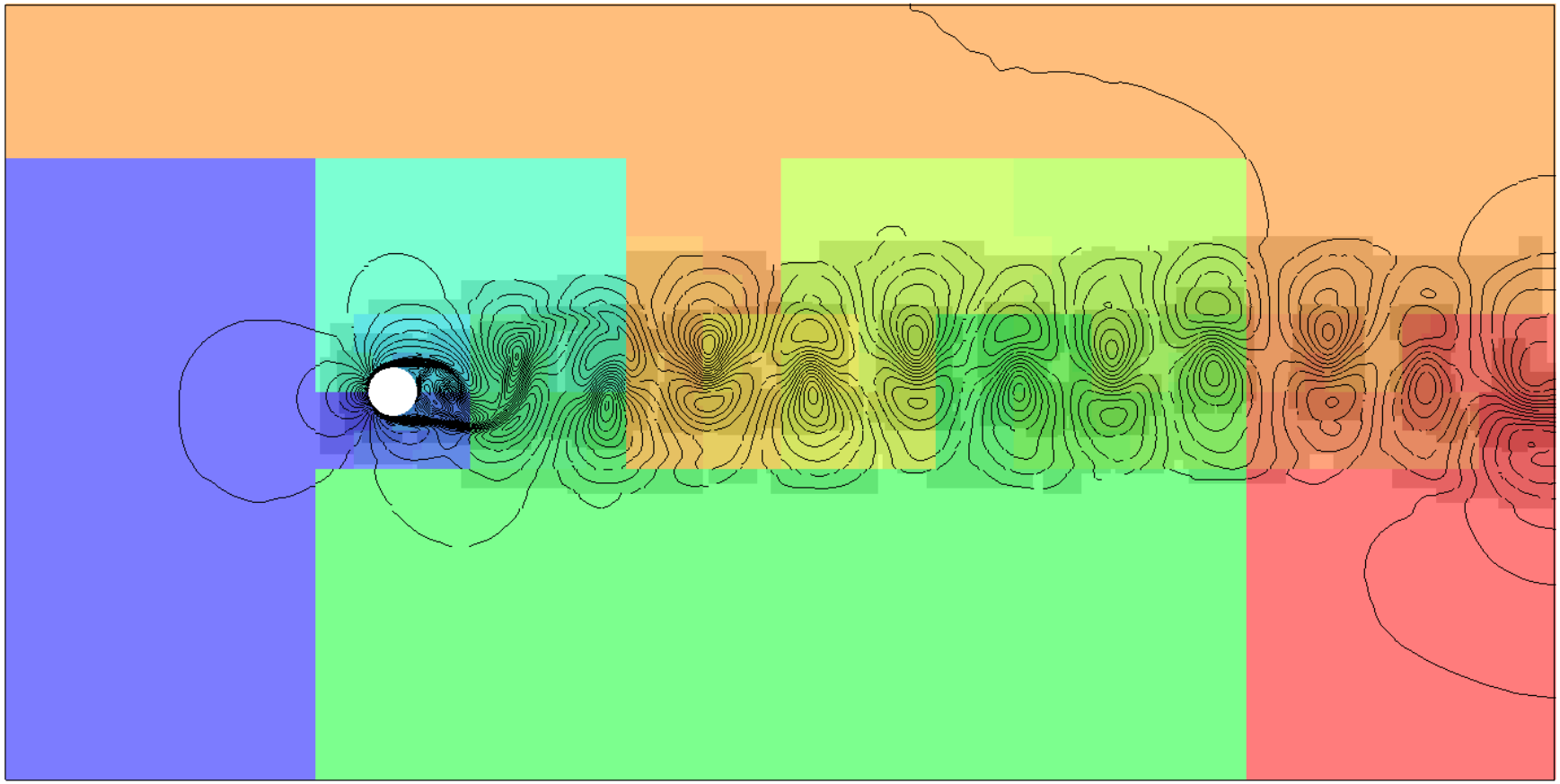
Isolines on refinement and distribution to processors

Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors

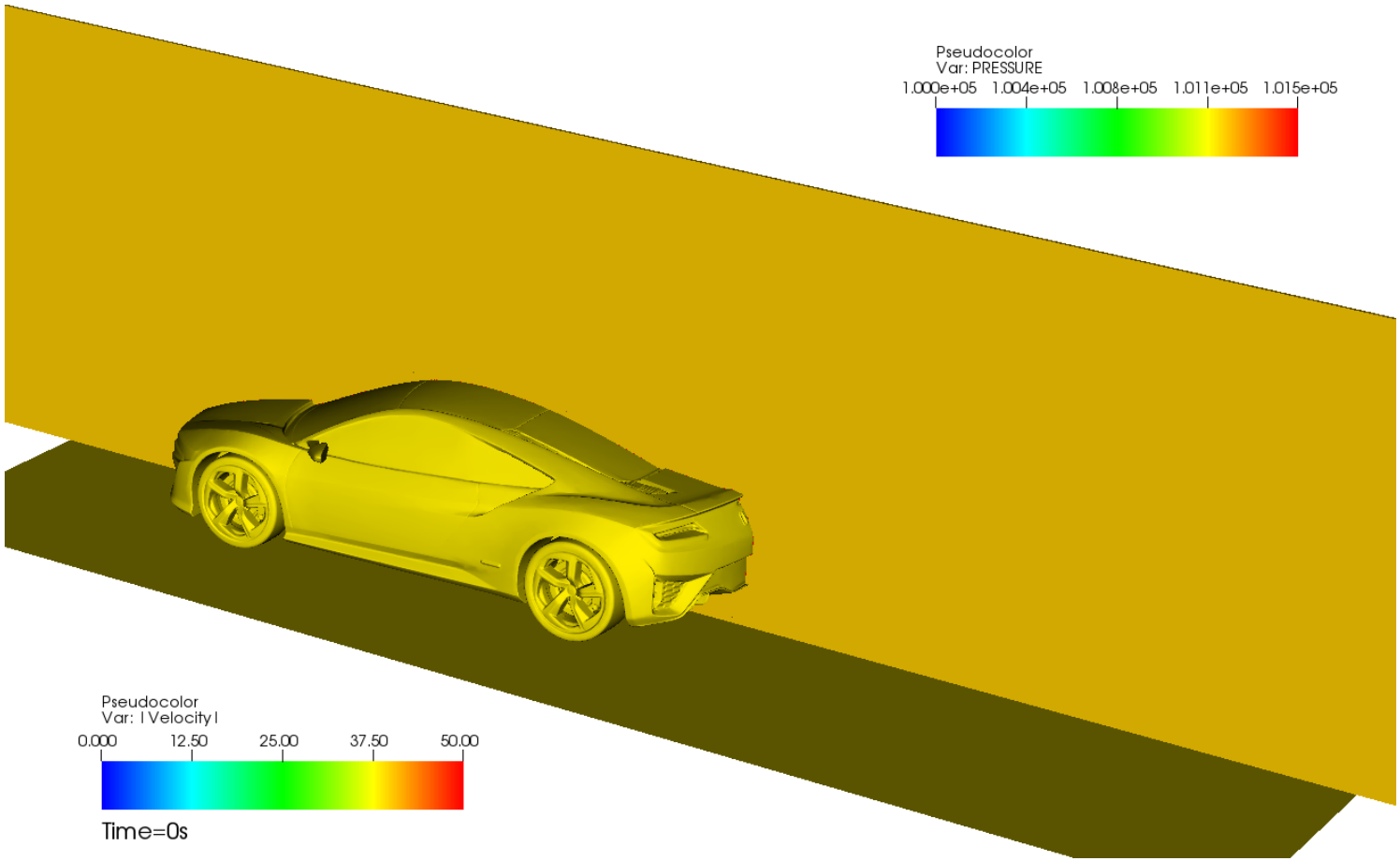
Flow over cylinder in 2d - $\mathrm{Re} = 300$, $u = 0.2415\,\mathrm{m/s}$

Isolines on refinement and distribution to processors



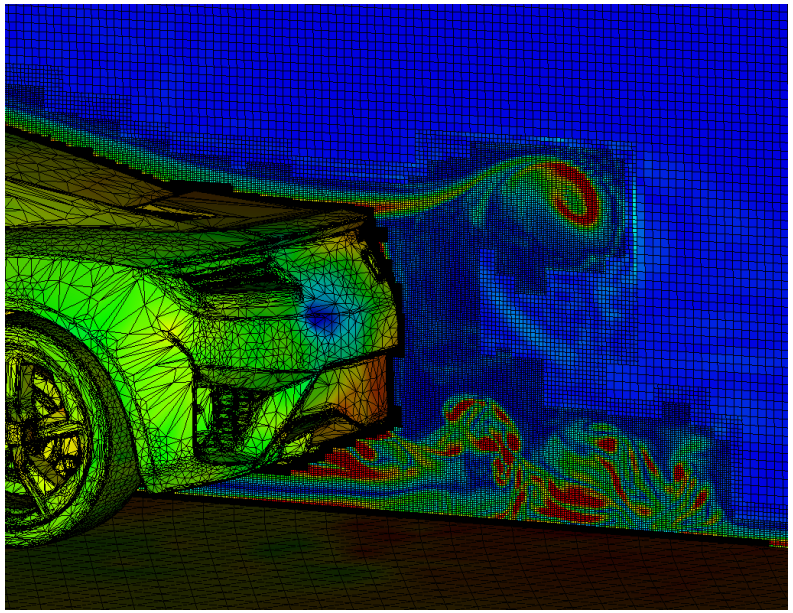
Wind tunnel simulation of a prototype car

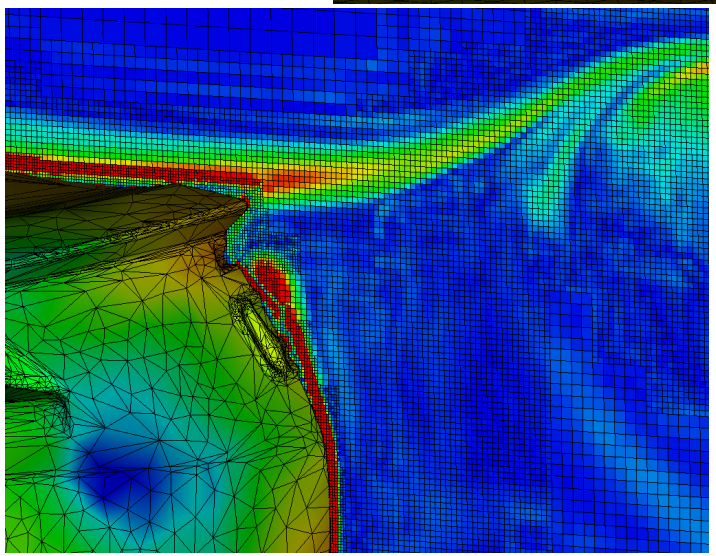
Fluid velocity and pressure on vehicle

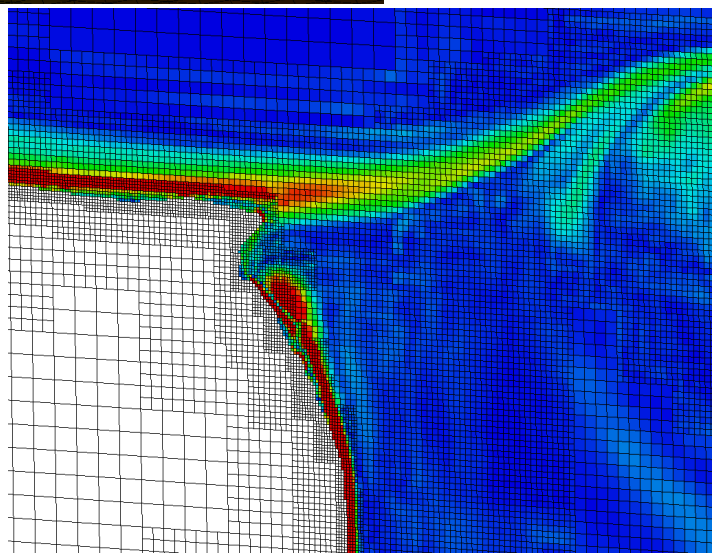


- Inflow 40 $\mathrm{m/s}$. LES model active. Characteristic boundary conditions.
- To $t = 0.5\,\mathrm{s}$ ($\sim$ 4 characteristic lengths) with 31,416 time steps on finest level in $\sim 37\,\mathrm{h}$ on 200 cores (7389 h CPU). Channel: $15\,\mathrm{m} \times 5\,\mathrm{m} \times 3.3\,\mathrm{m}$

Mesh adaptation

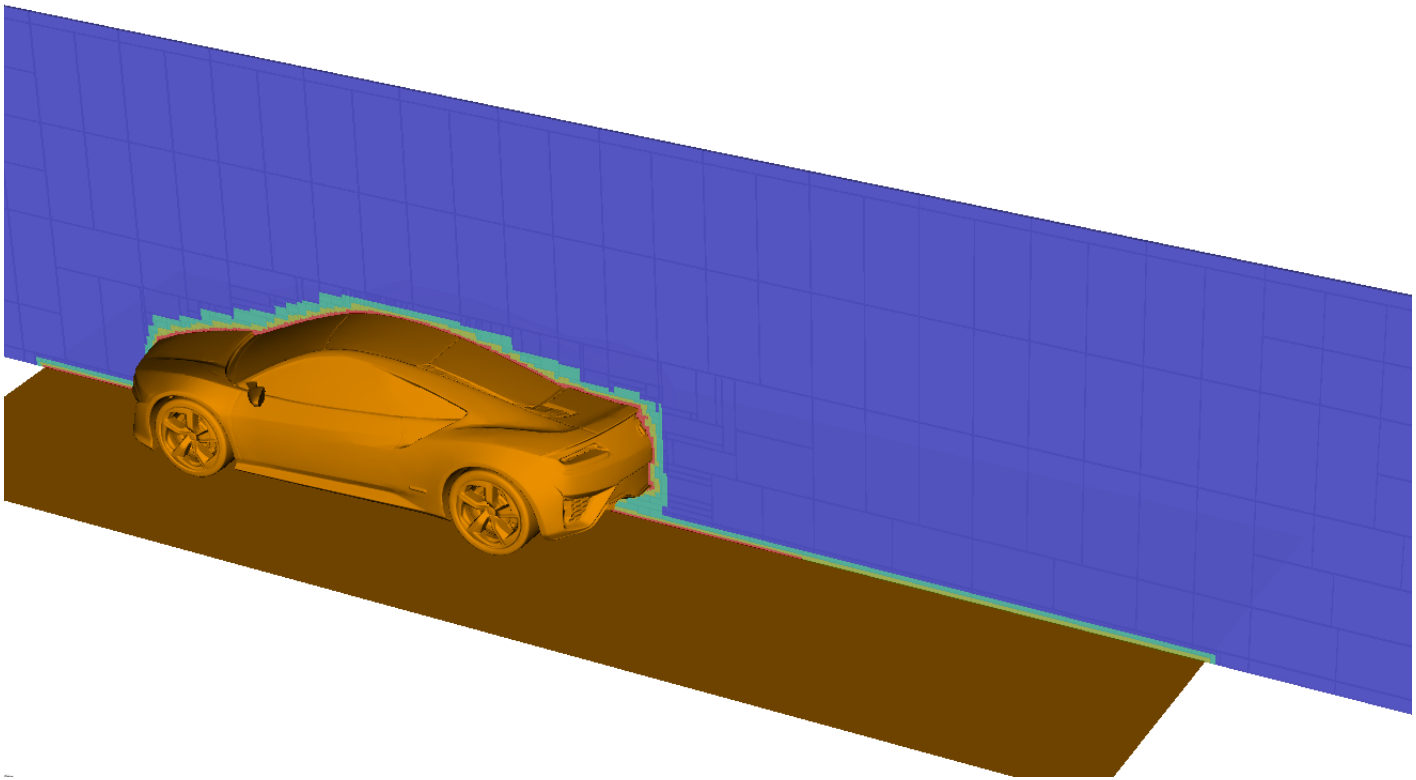






Mesh adaptation

Used refinement blocks and levels (indicated by color)

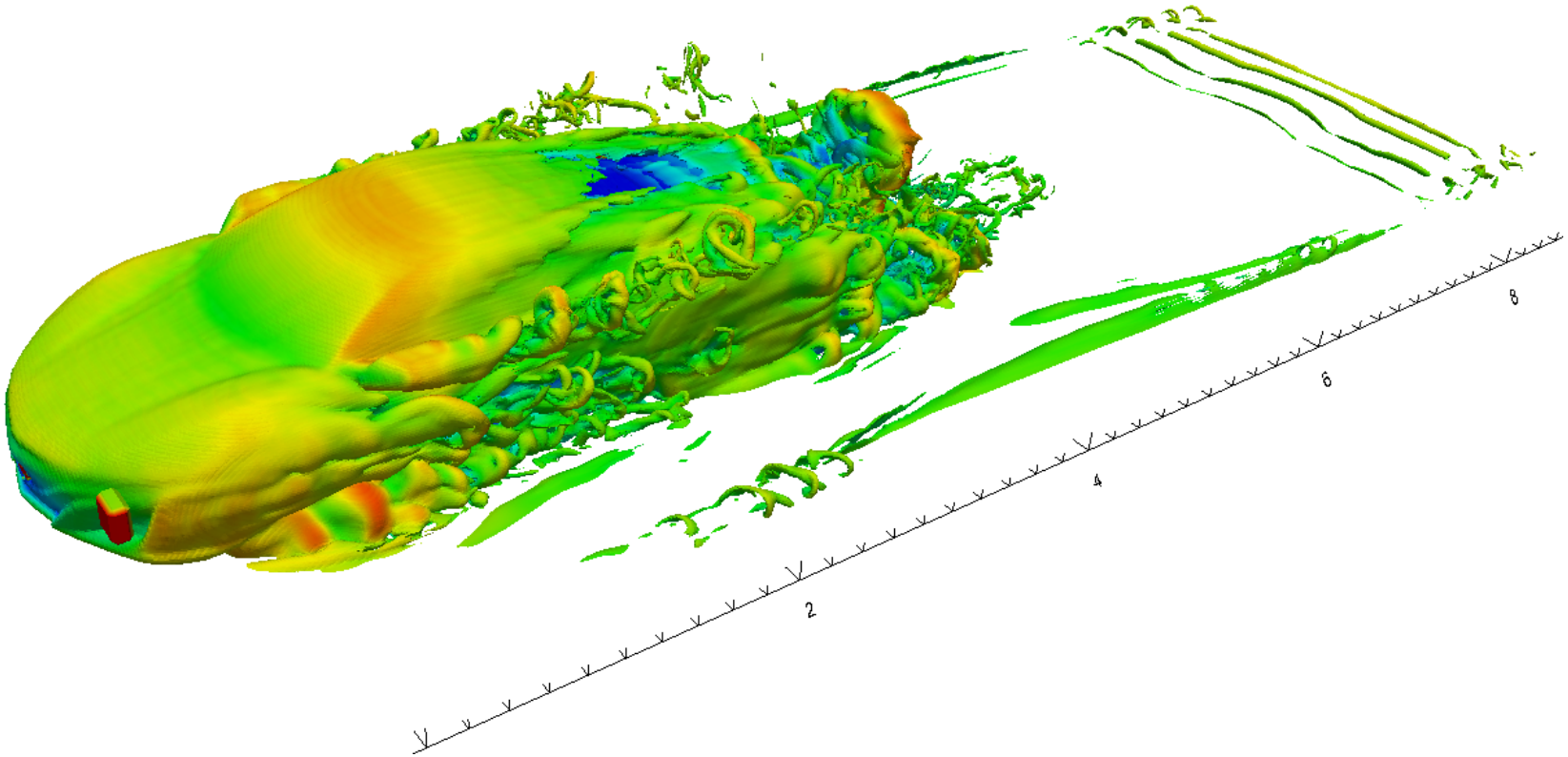


- SAMR base grid $600 \times 200 \times 132$ cells, $r_{1,2,3} = 2$ yielding finest resolution of $\Delta x = 3.125\,\mathrm{mm}$
- Adaptation based on level set and scaled gradient of magnitude of vorticity vector
- 236M cells vs. 8.1 billion (uniform) at $t = 0.4075\,\mathrm{s}$

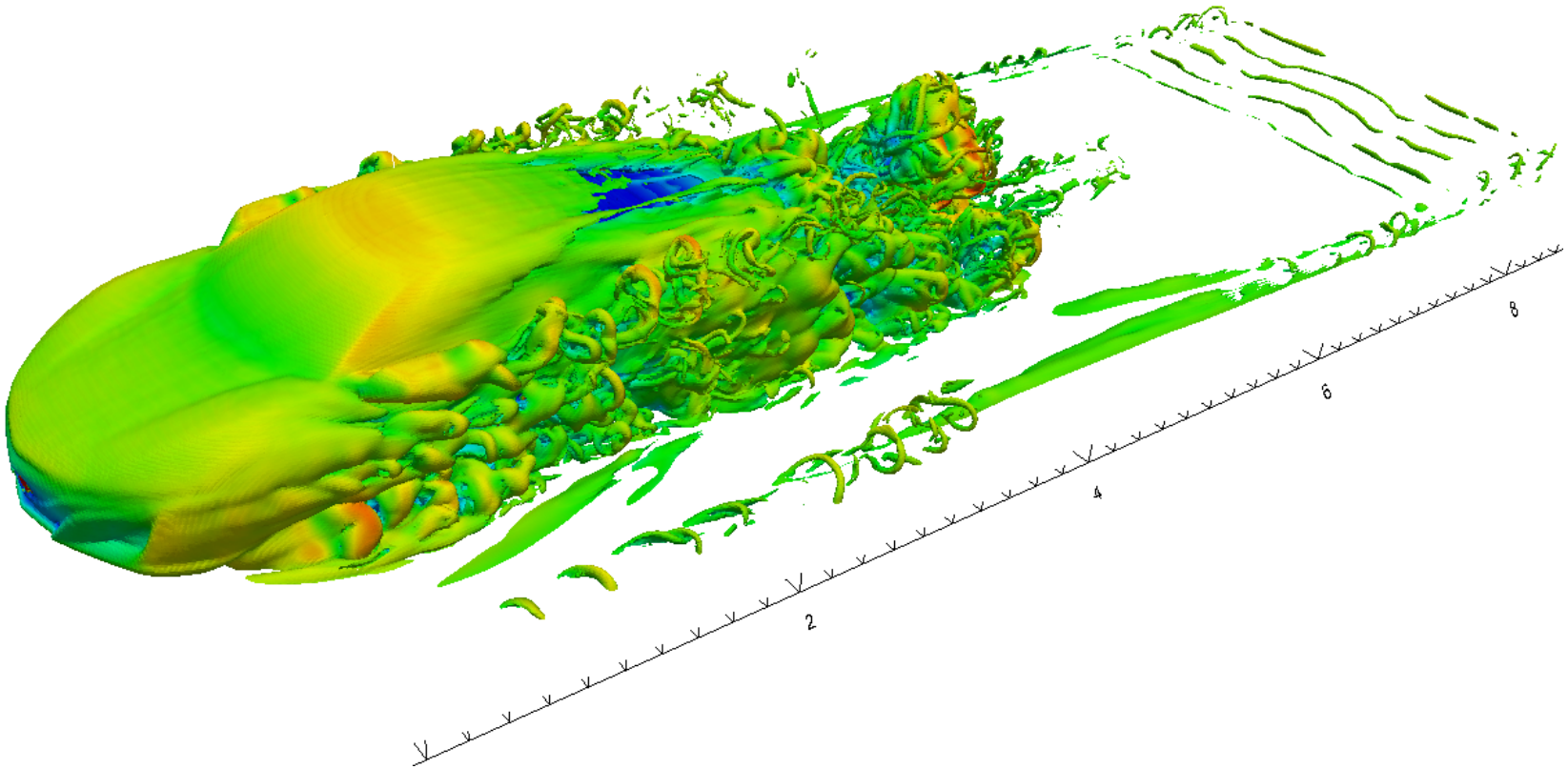
Refinement at $t = 0.4075\,\mathrm{s}$

Level	Grids	Cells
0	11,605	15,840,000
1	11,513	23,646,984
2	31,382	144,447,872
3	21,221	52,388,336

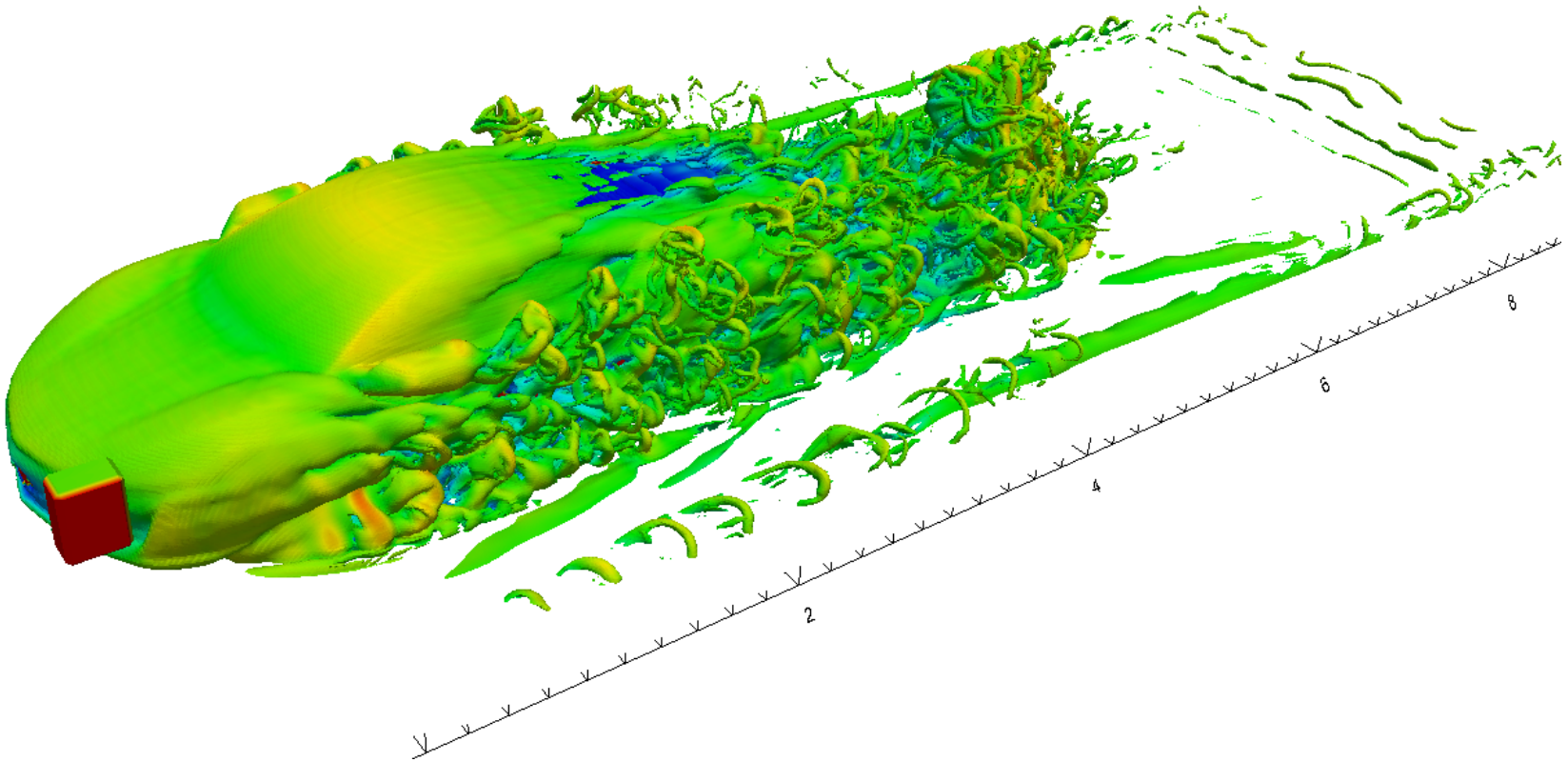
Turbulent structures



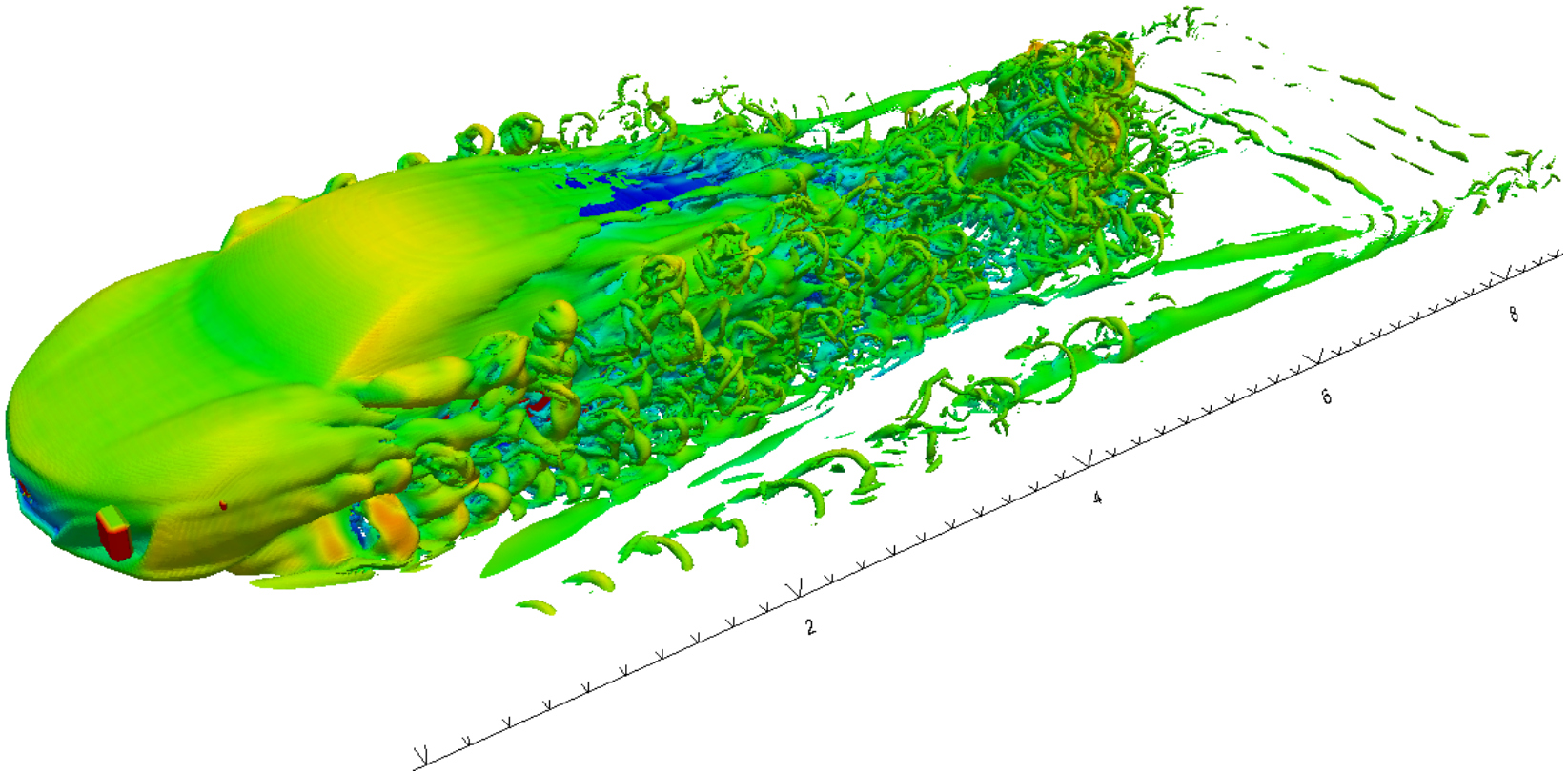
Turbulent structures



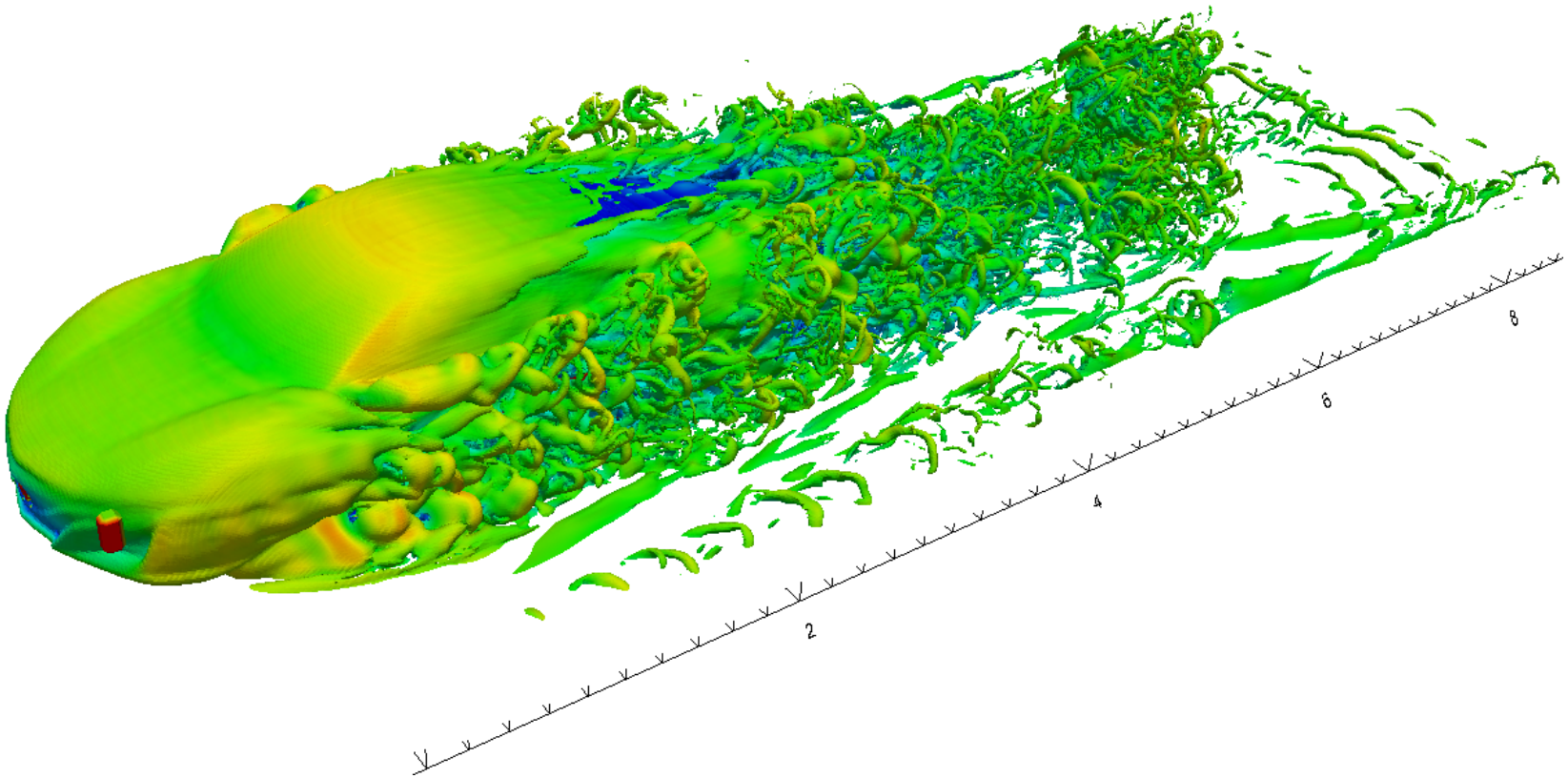
Turbulent structures



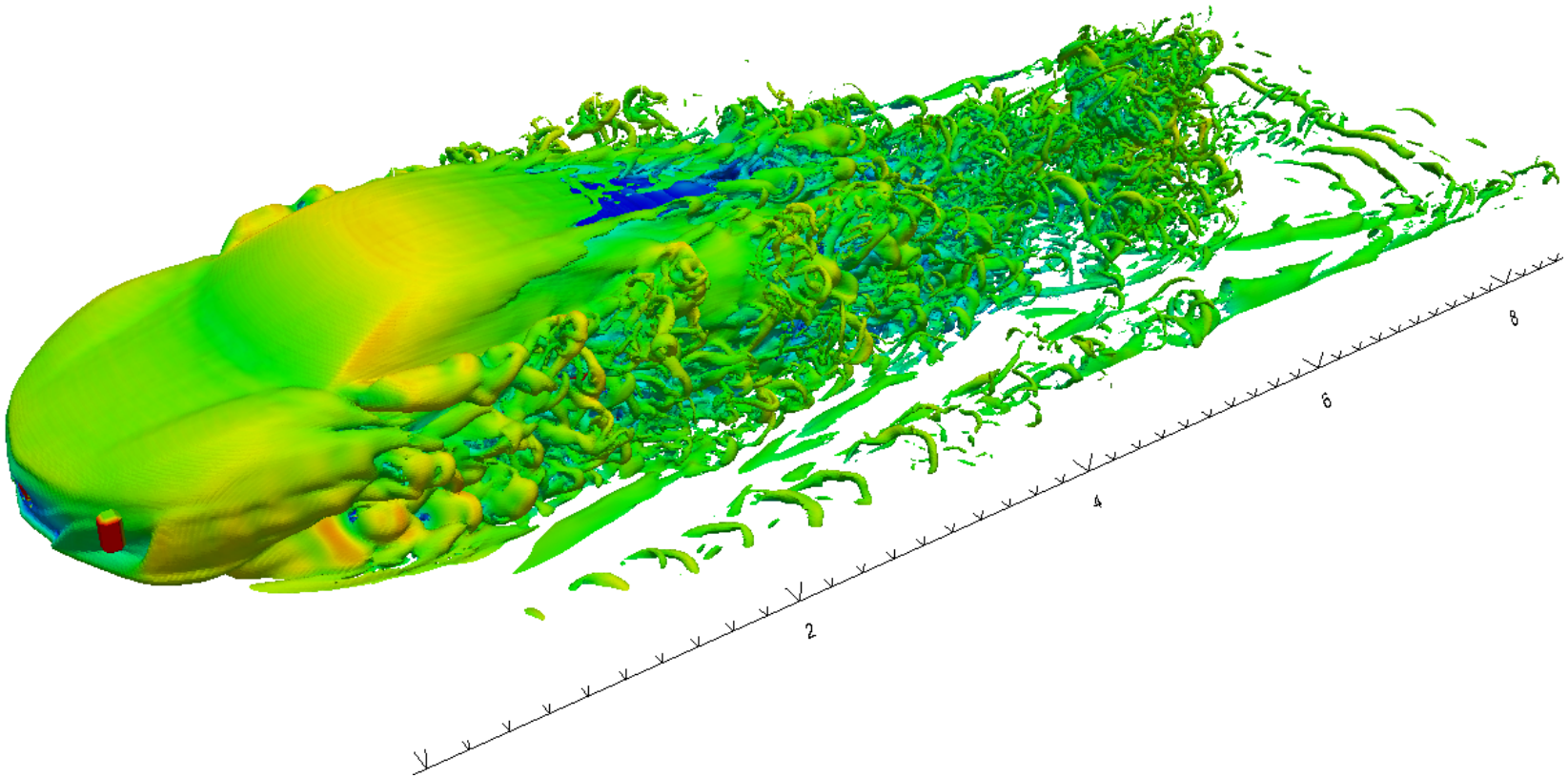
Turbulent structures



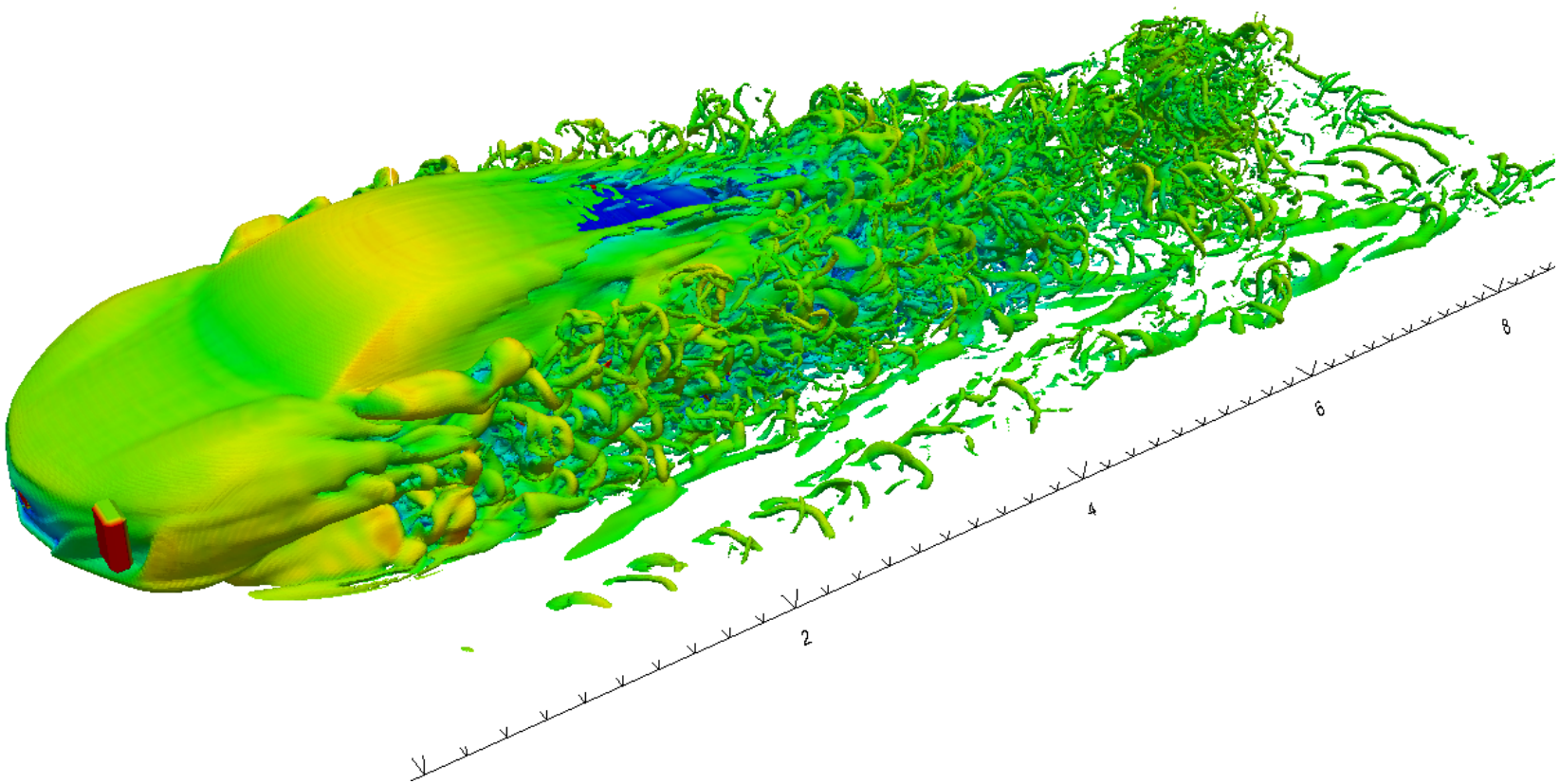
Turbulent structures



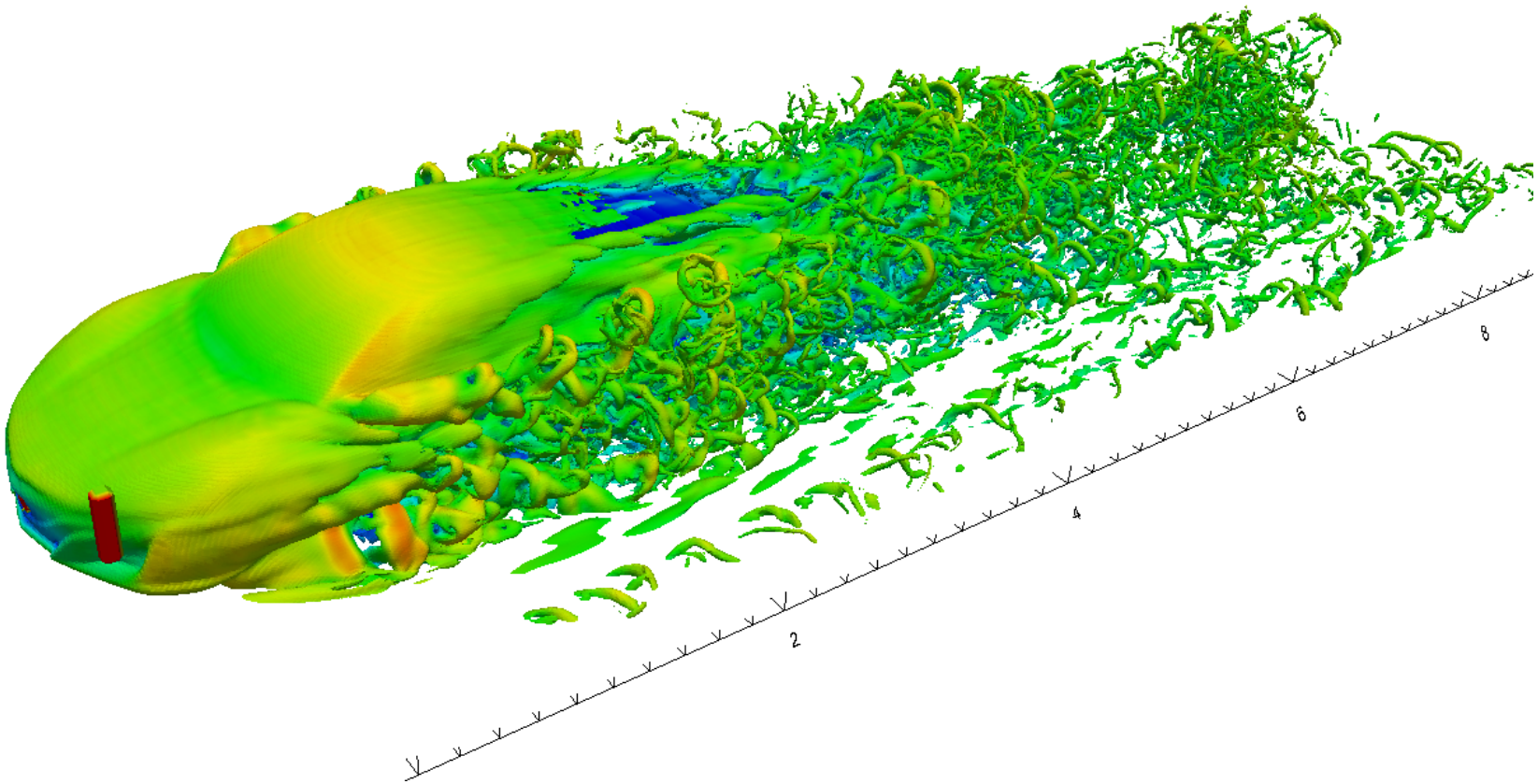
Turbulent structures



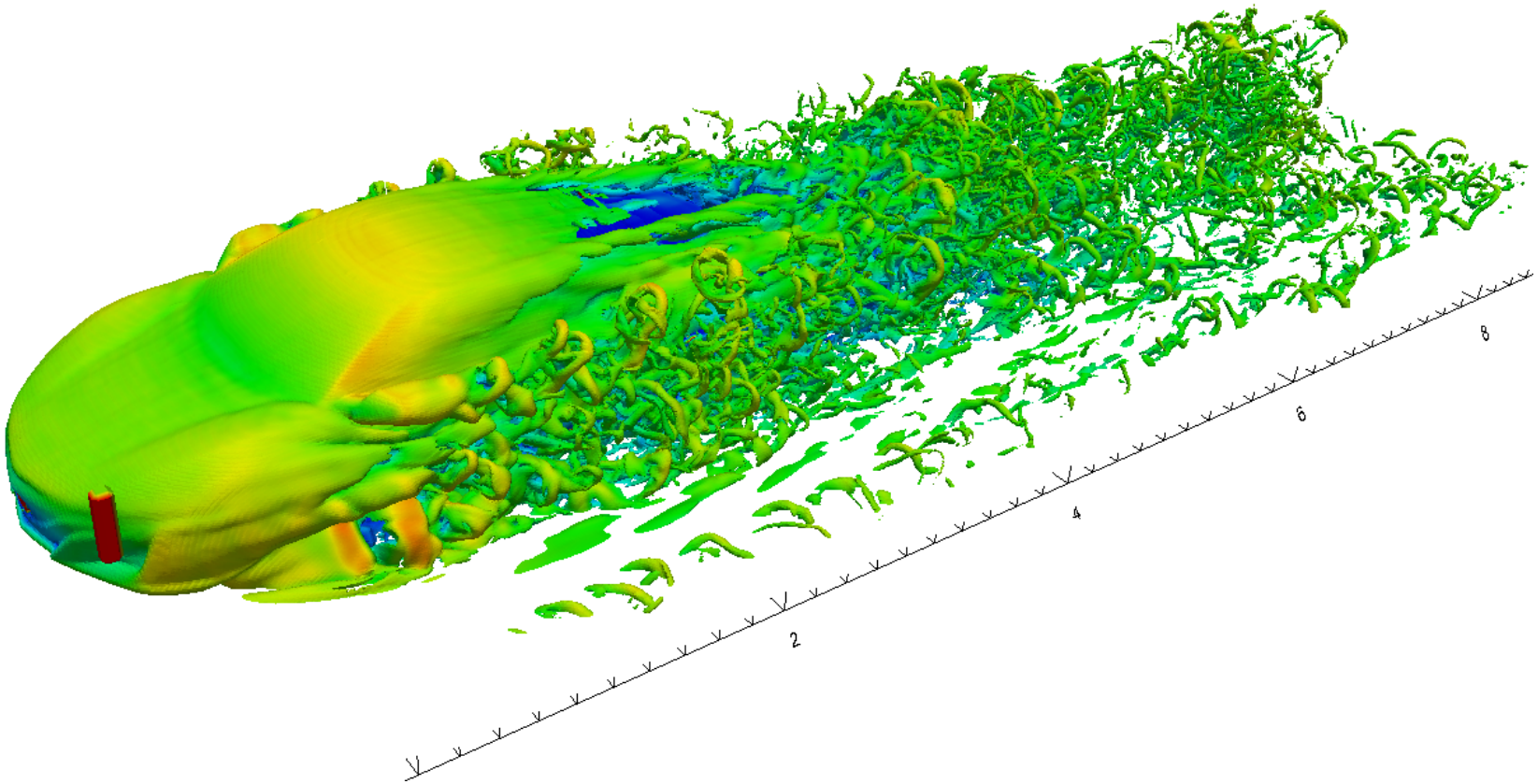
Turbulent structures



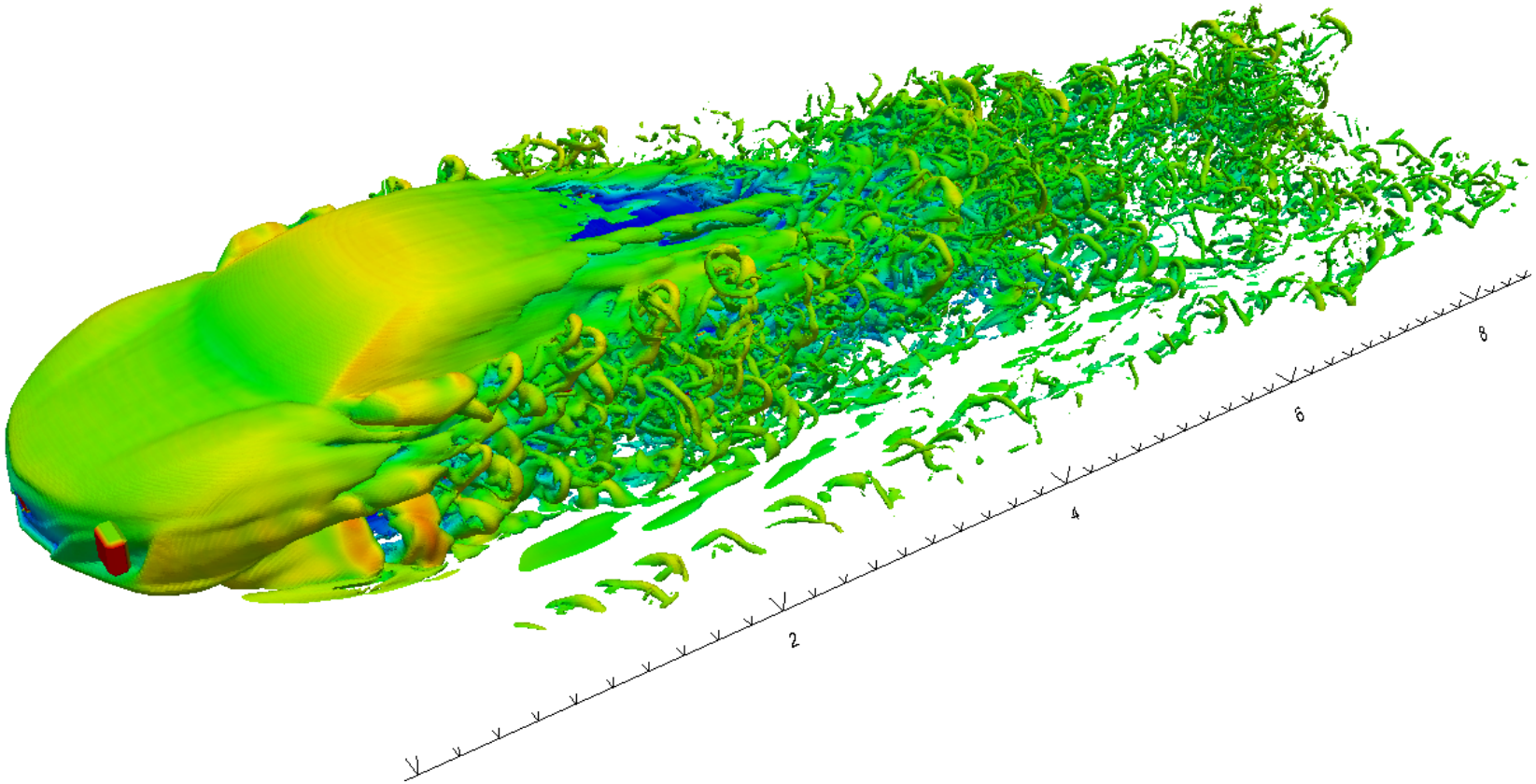
Turbulent structures



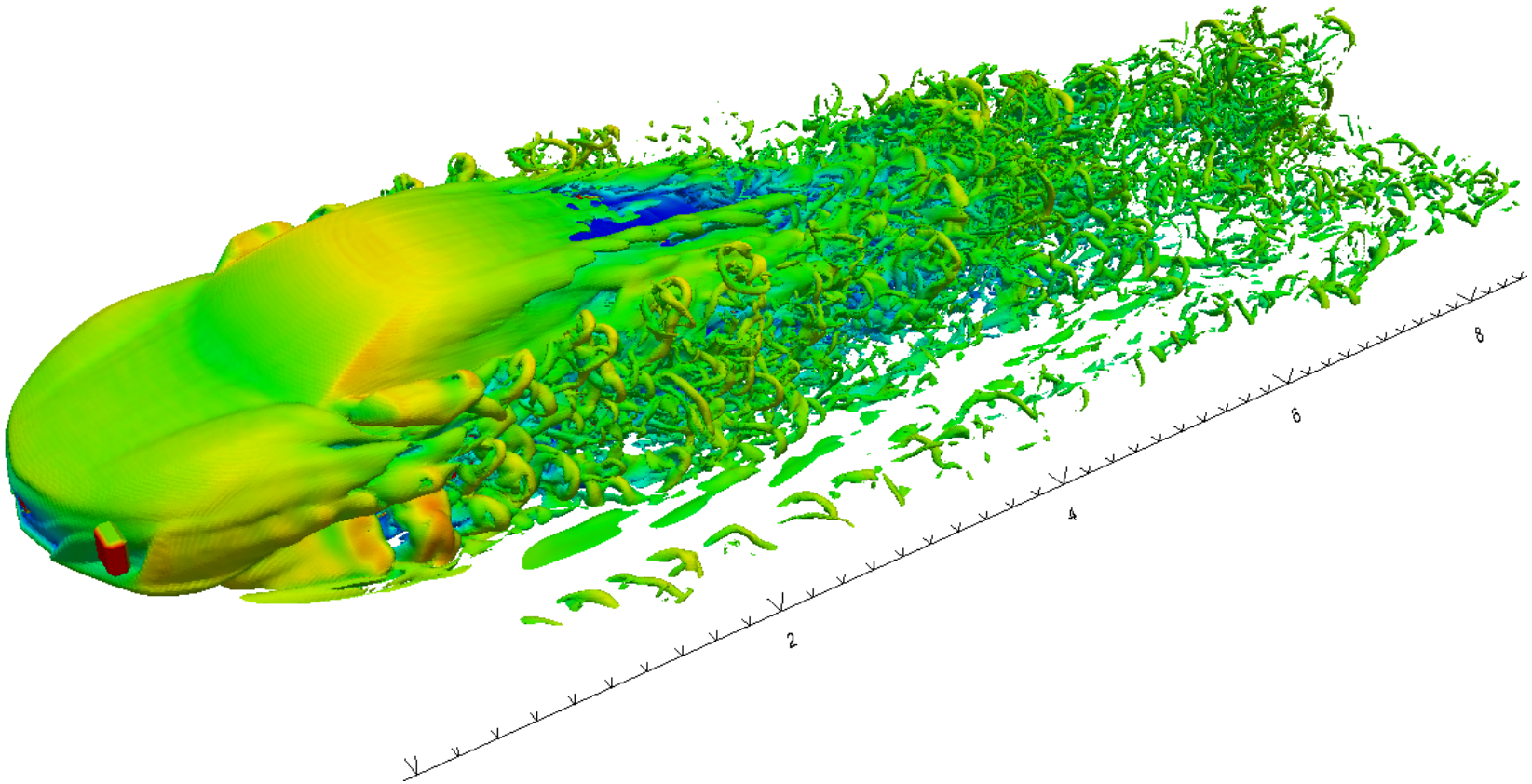
Turbulent structures



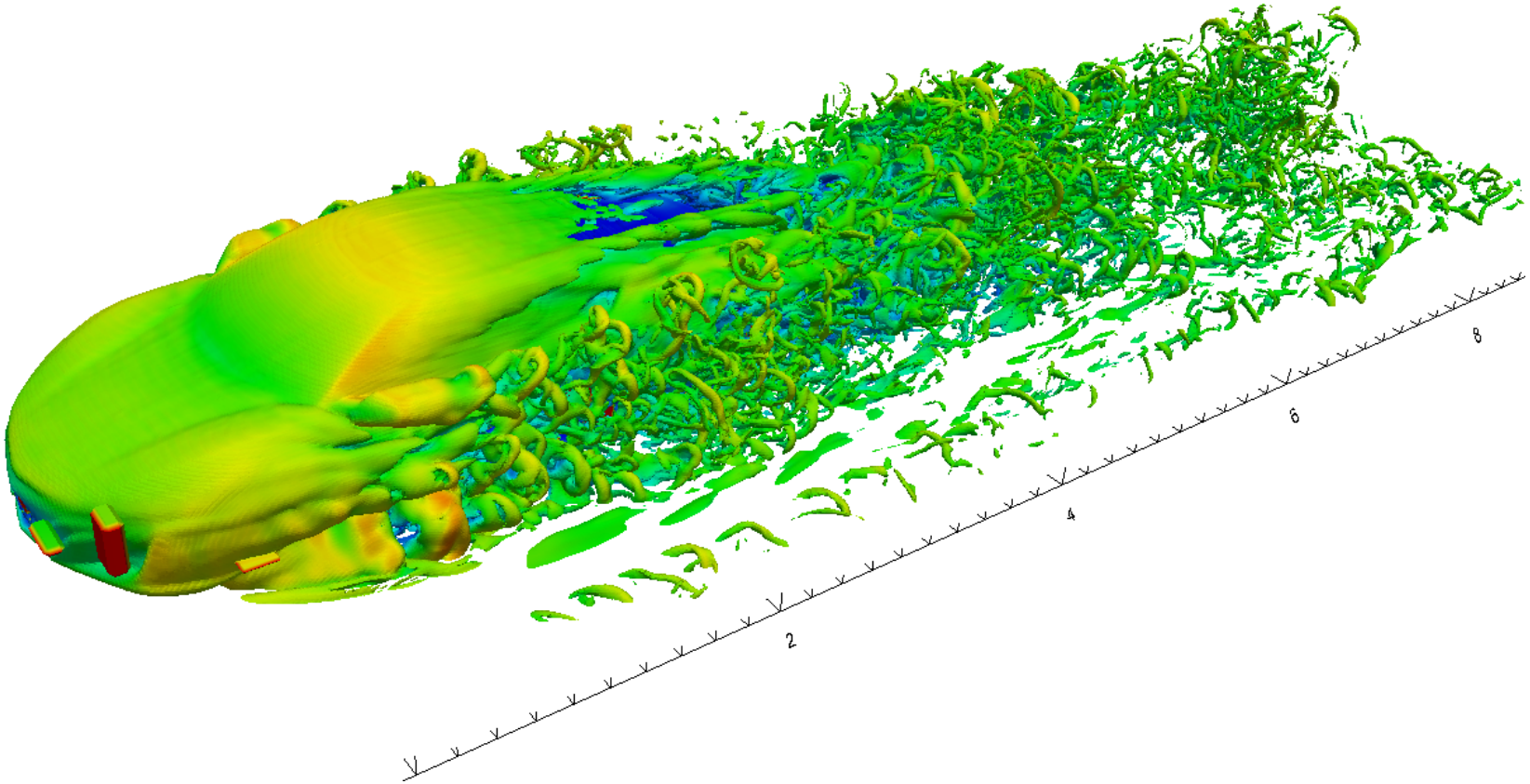
Turbulent structures



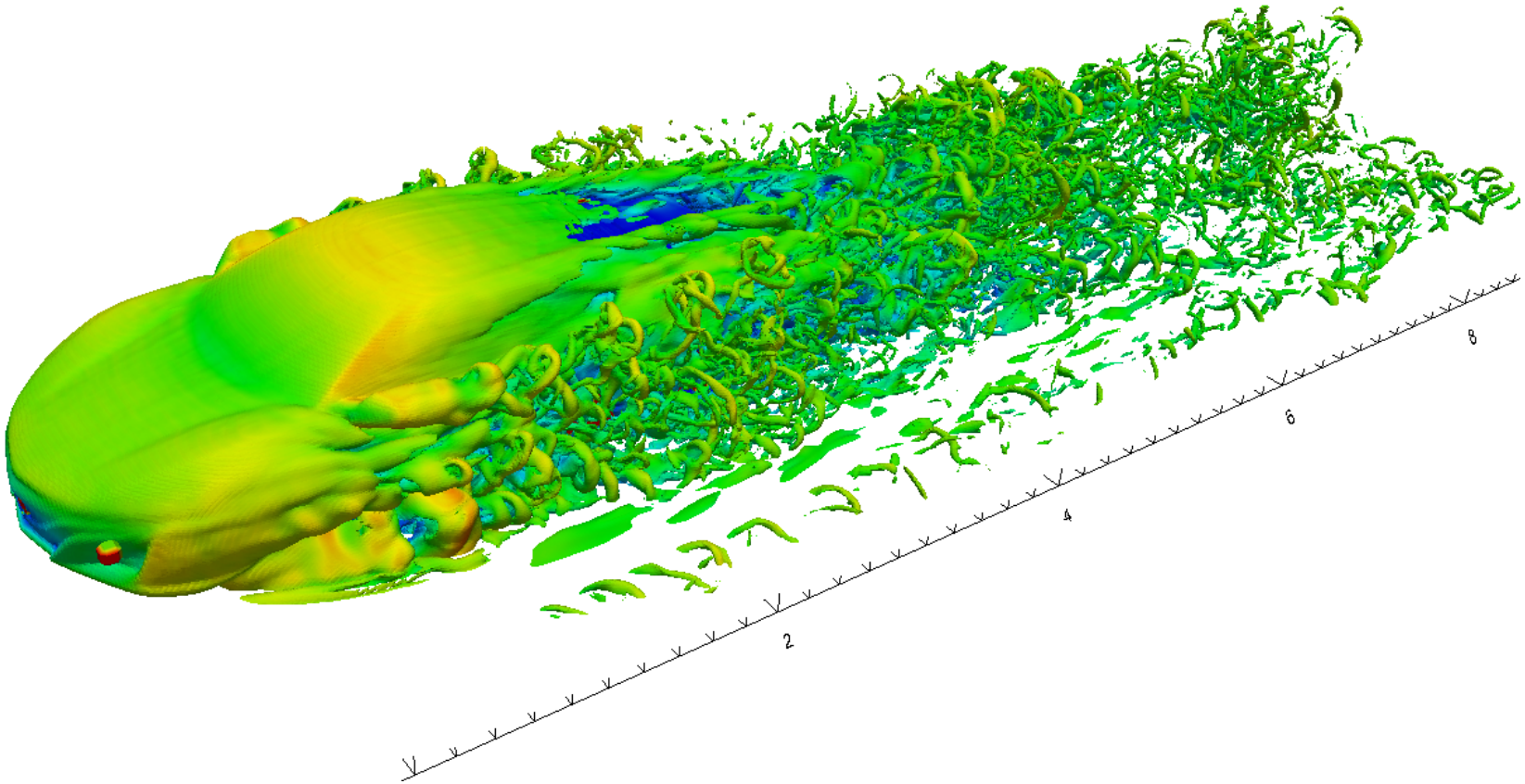
Turbulent structures



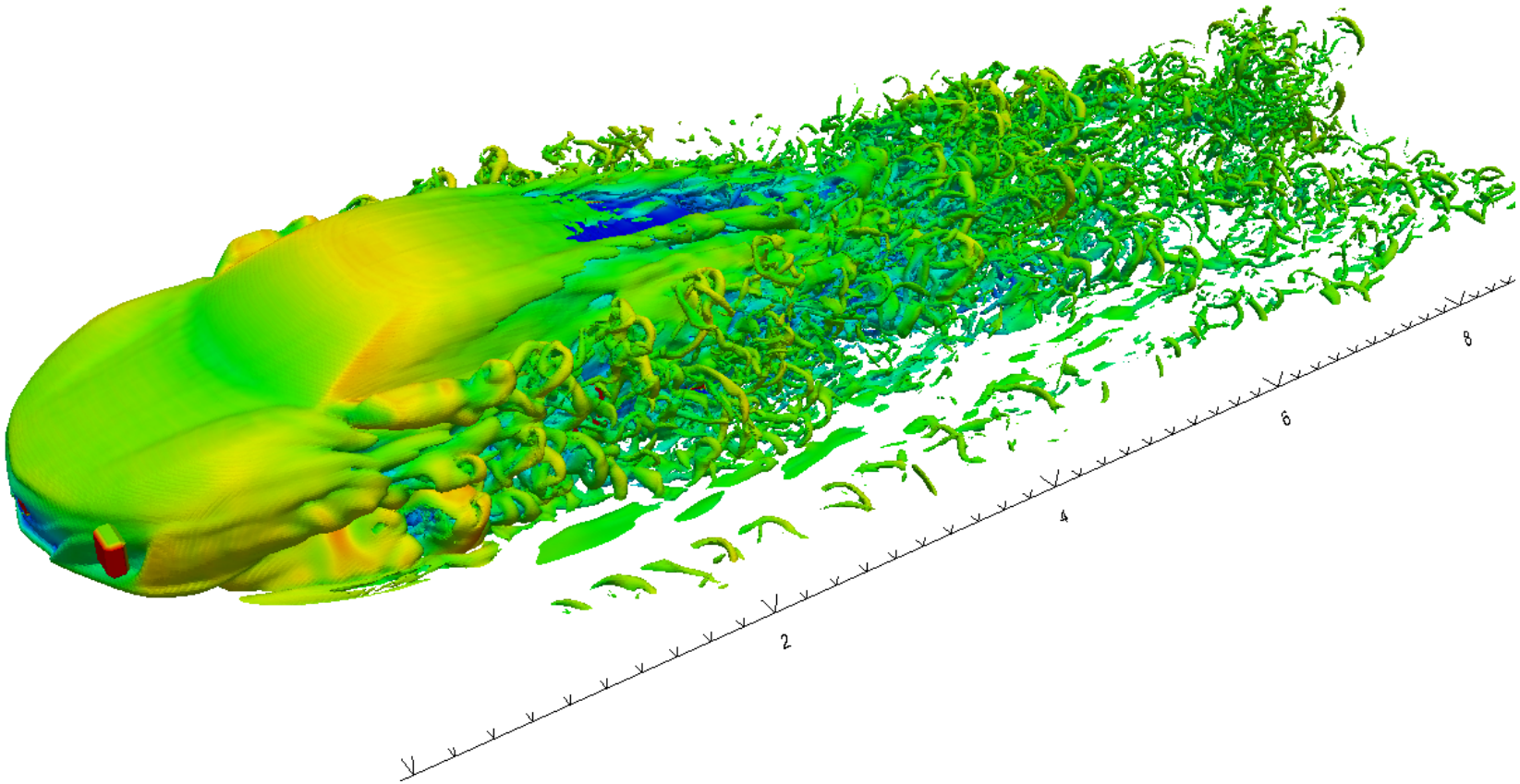
Turbulent structures



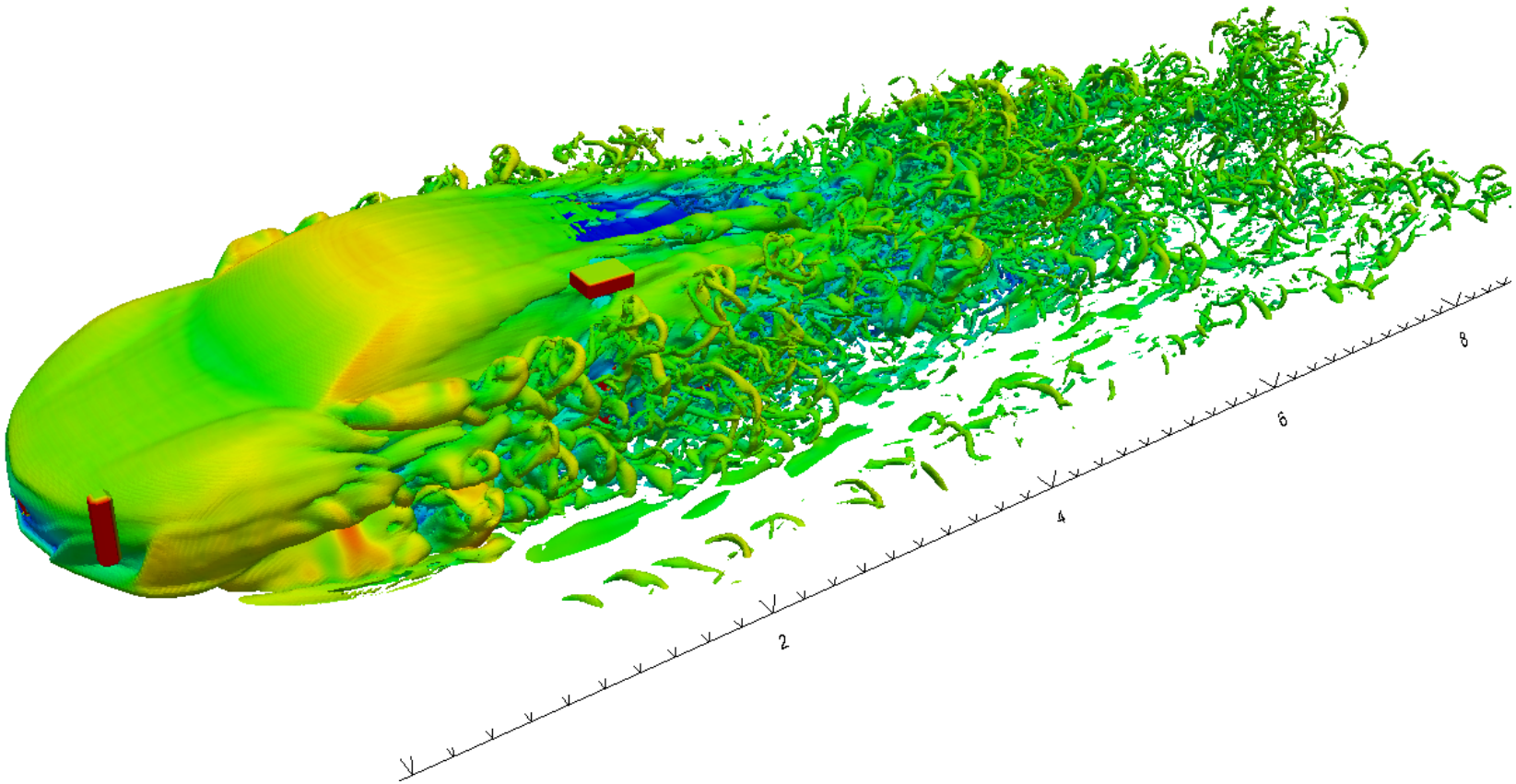
Turbulent structures



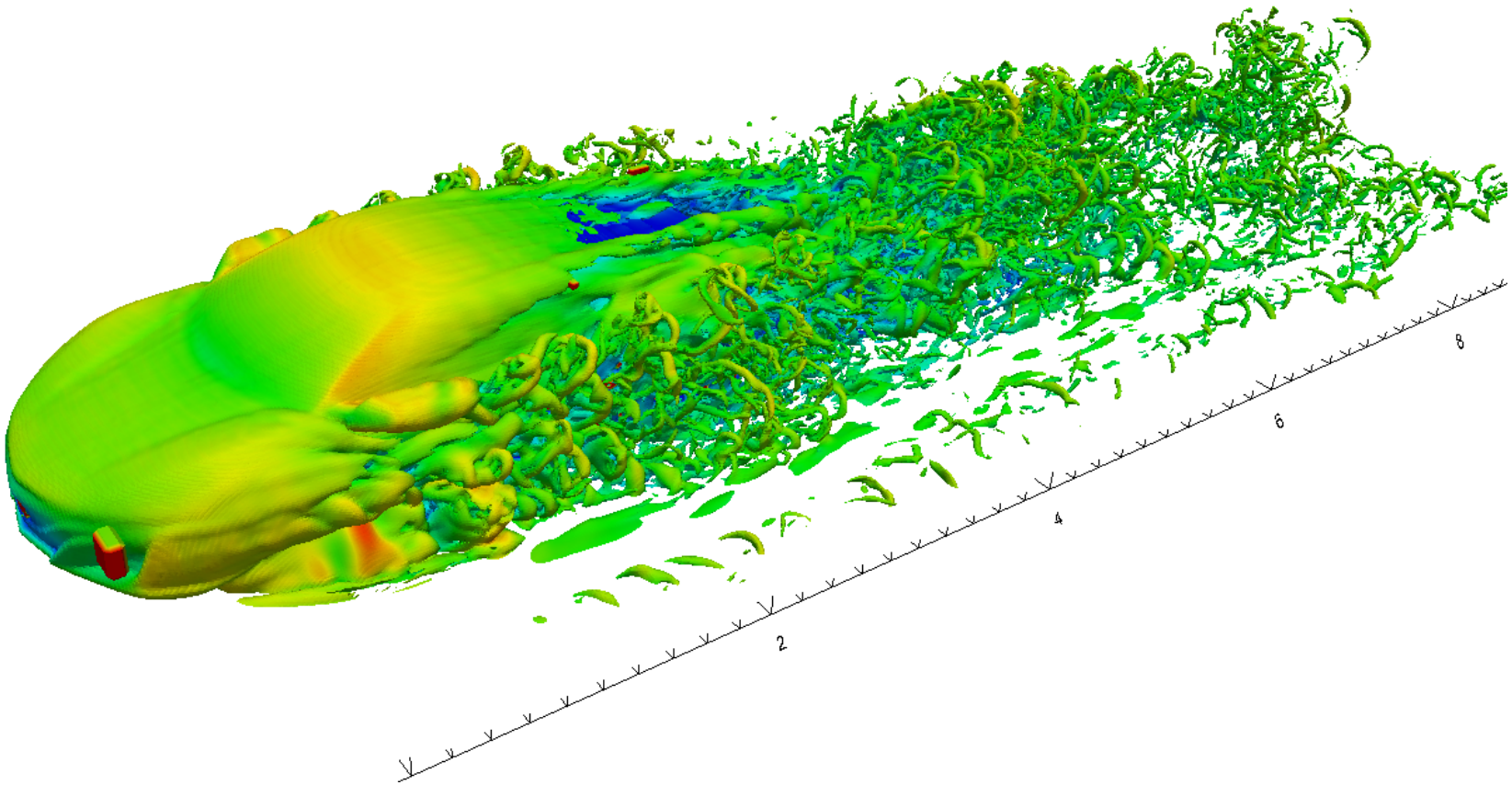
Turbulent structures



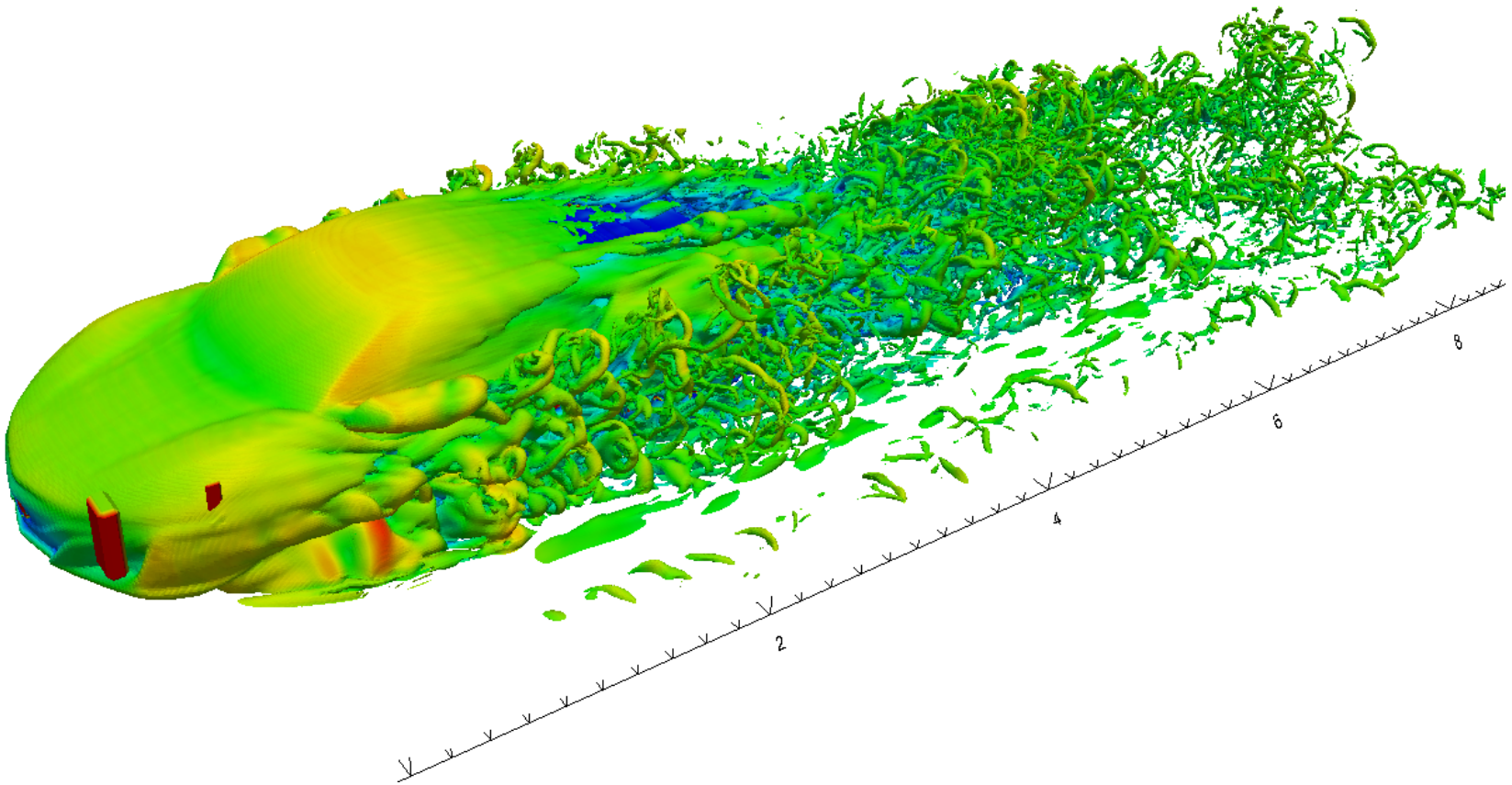
Turbulent structures



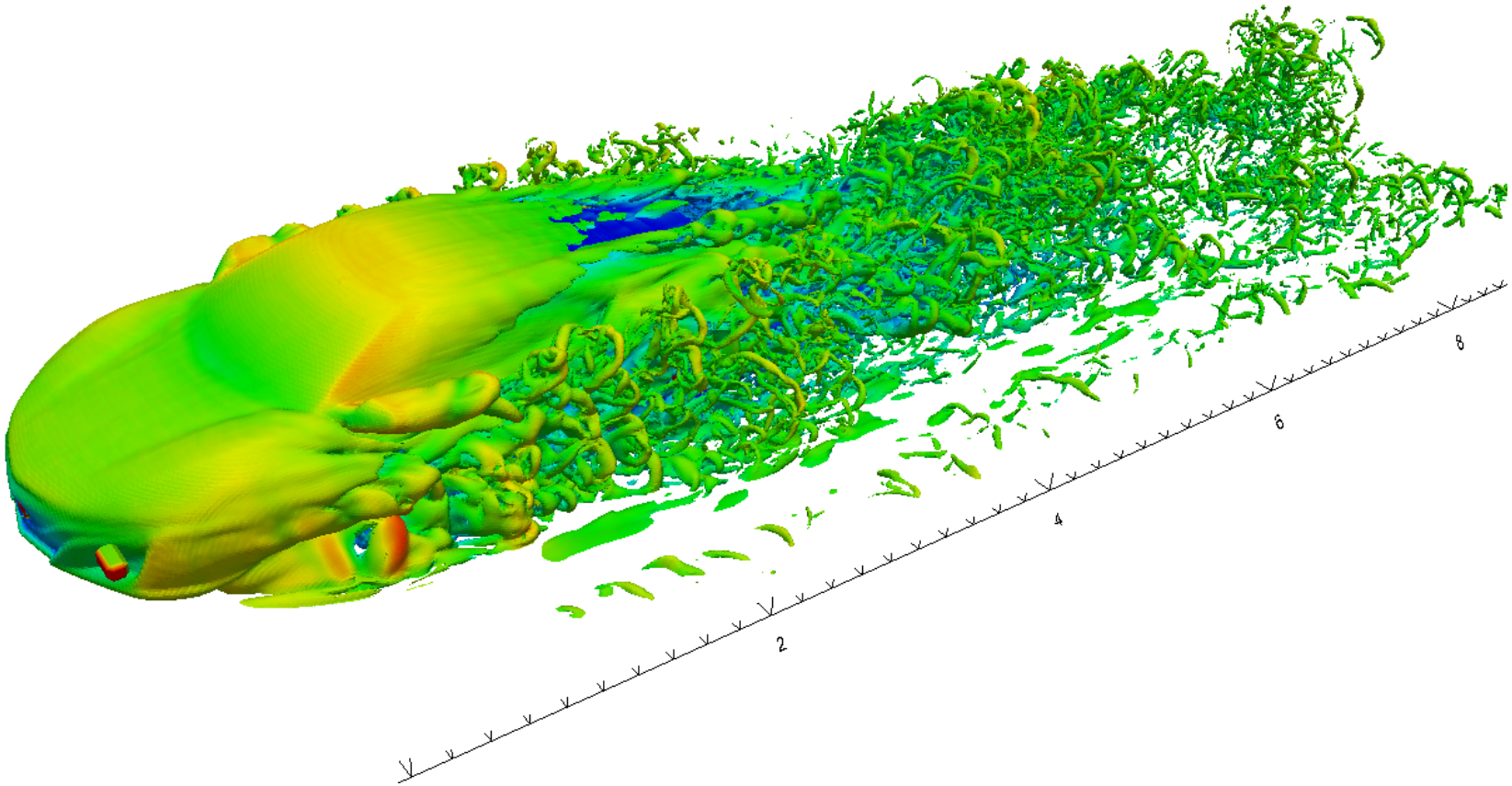
Turbulent structures



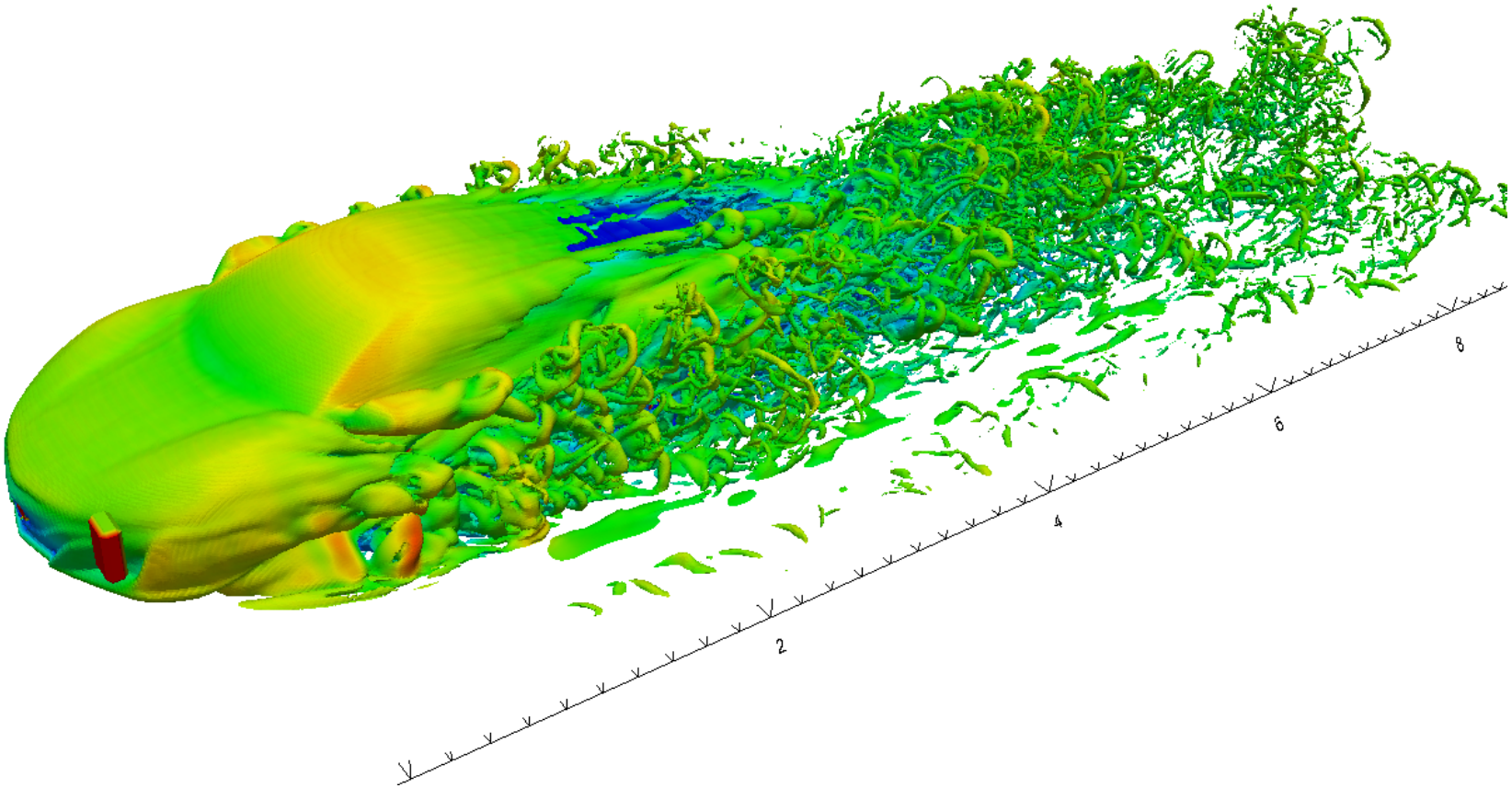
Turbulent structures



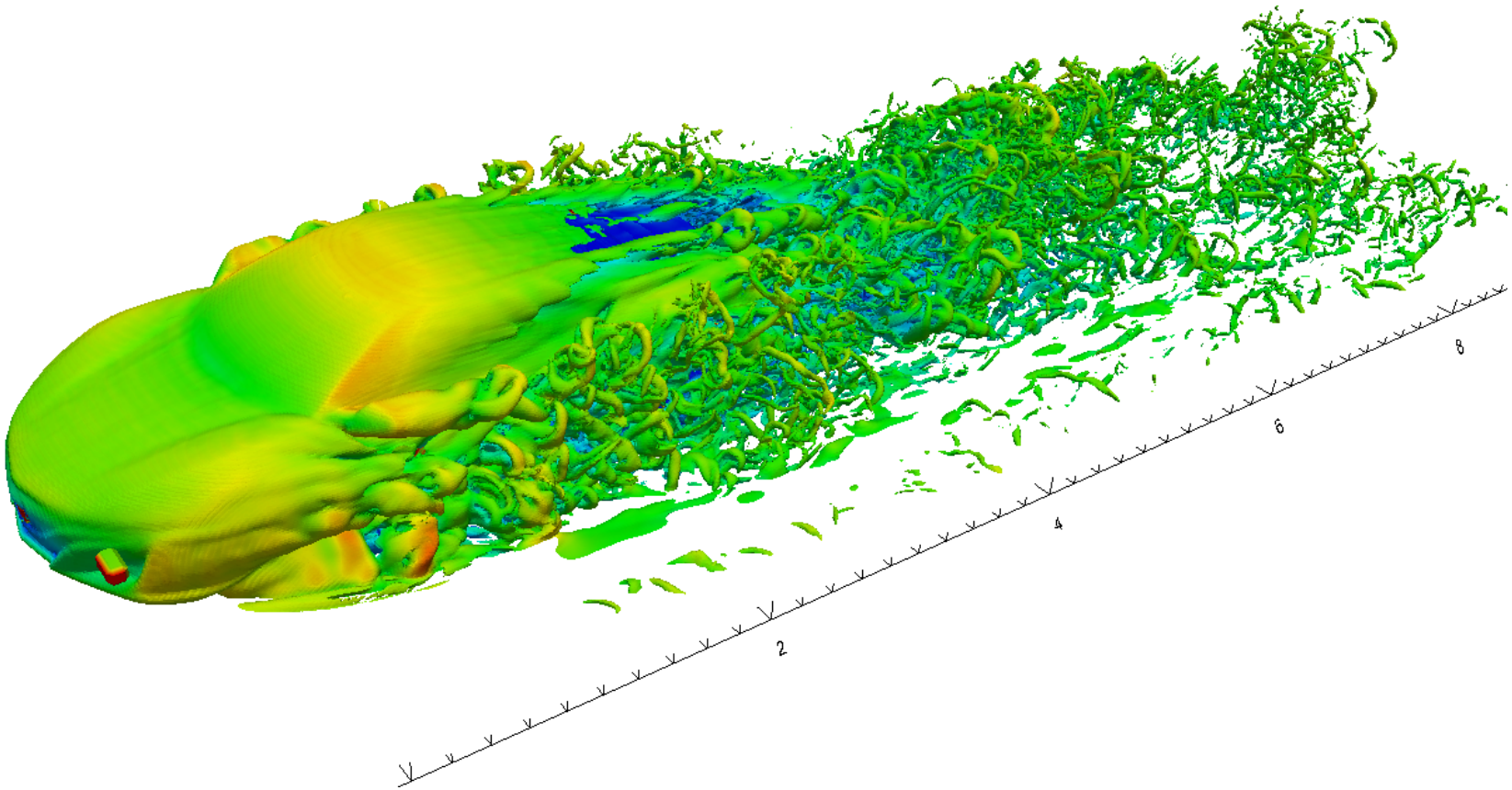
Turbulent structures



Turbulent structures

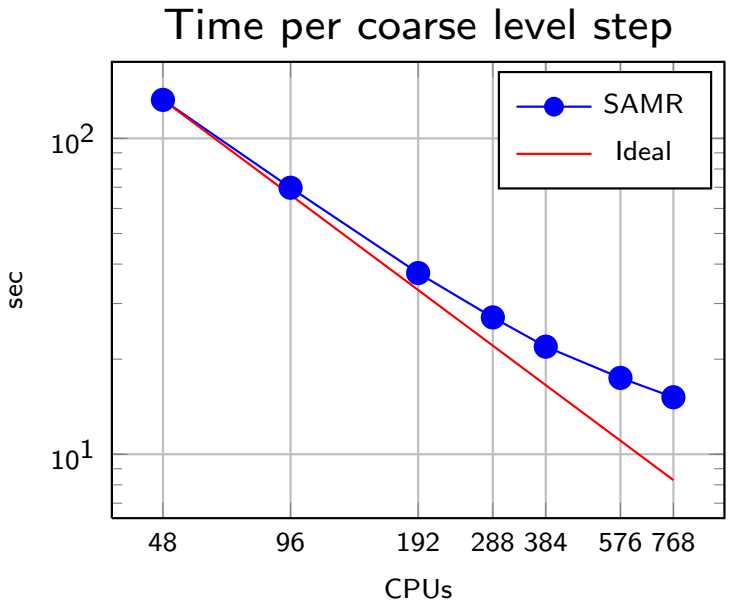


Turbulent structures



Strong scalability test (1:25 train)

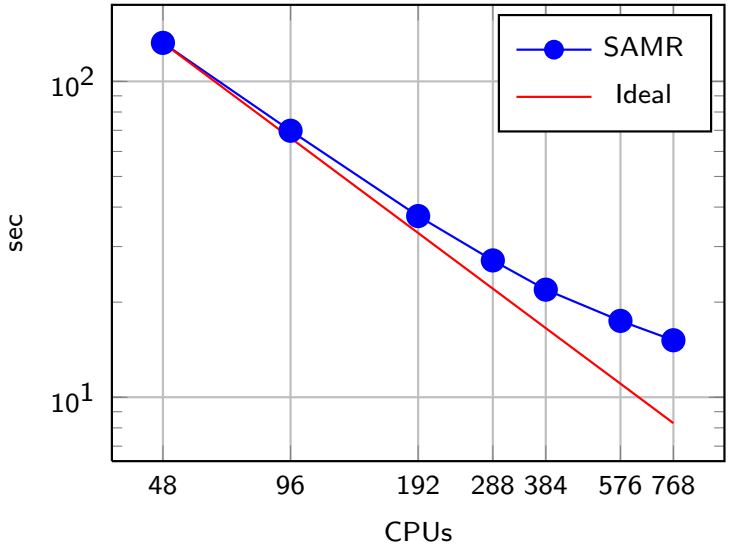
- Computation is restarted from disk checkpoint at $t = 0.526408\,\mathrm{s}$ from 96 core run.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).



Strong scalability test (1:25 train)

- Computation is restarted from disk checkpoint at $t = 0.526408\,\mathrm{s}$ from 96 core run.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).

Time per coarse level step



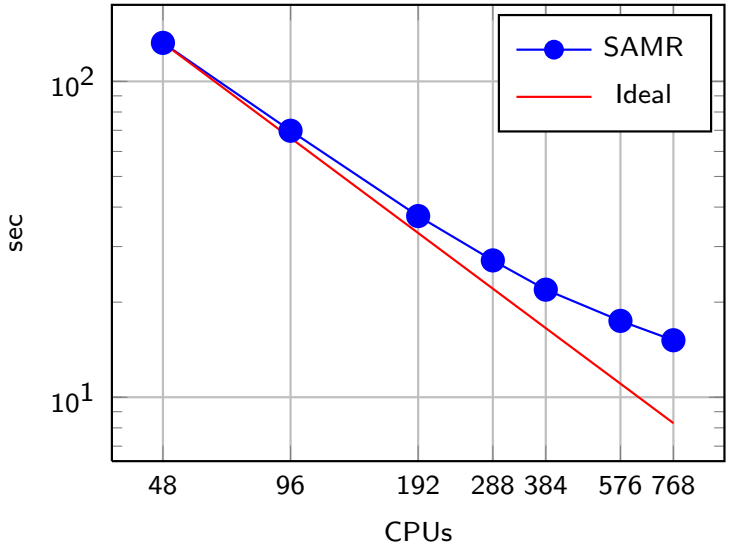
Time in % spent in main operations

Cores	48	96	192	288	384	576	768
Time per step	132.43s	69.79s	37.47s	27.12s	21.91s	17.45s	15.15s
Par. Efficiency	100.0	94.88	88.36	81.40	75.56	63.24	54.63
LBM Update	5.91	5.61	5.38	4.92	4.50	3.73	3.19
Regridding	15.44	12.02	11.38	10.92	10.02	8.94	8.24
Partitioning	4.16	2.43	1.16	1.02	1.04	1.16	1.34
Interpolation	3.76	3.53	3.33	3.05	2.83	2.37	2.06
Sync Boundaries	54.71	59.35	59.73	56.95	54.54	52.01	51.19
Sync Fixup	9.10	10.41	12.25	16.62	20.77	26.17	28.87
Level set	0.78	0.93	1.21	1.37	1.45	1.48	1.47
Interp./Extrap.	3.87	3.81	3.76	3.49	3.26	2.75	2.39
Misc	2.27	1.91	1.79	1.67	1.58	1.38	1.25

Strong scalability test (1:25 train)

- Computation is restarted from disk checkpoint at $t = 0.526408\,\mathrm{s}$ from 96 core run.
- Time for initial re-partitioning removed from benchmark.
- 200 coarse level time steps computed.
- Regridding and re-partitioning every 2nd level-0 step.
- Computation starts with 51.8M cells (l3: 10.2M, l2: 15.3M, l1: 21.5M, l0= 4.8M) vs. 19.66 billion (uniform).
- Portions for parallel communication quite considerable (4 ghost cells still used).

Time per coarse level step

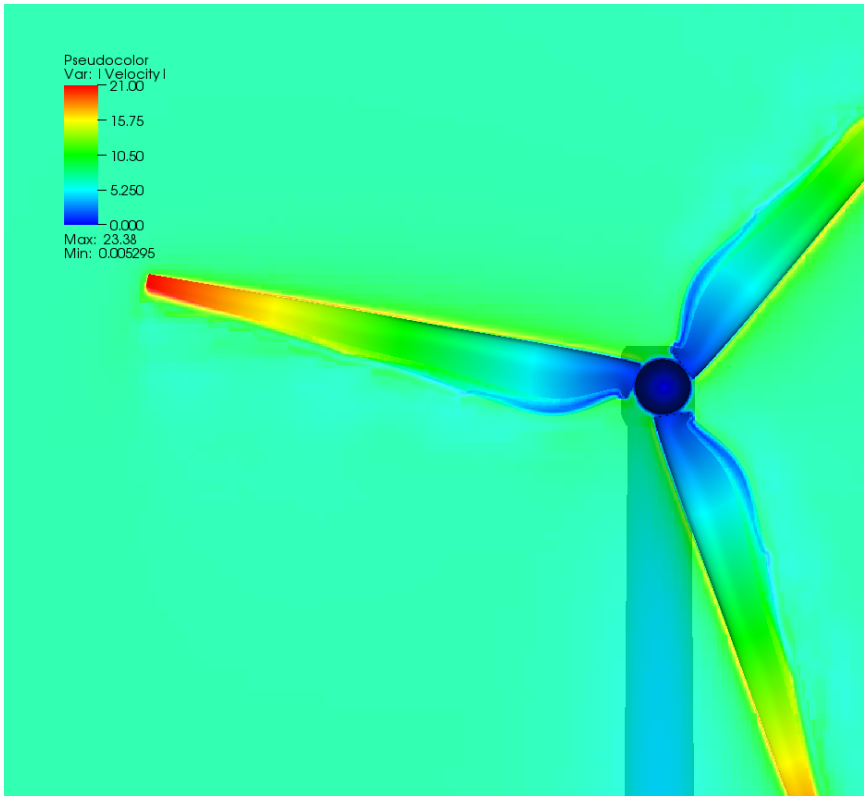


Time in % spent in main operations

Cores	48	96	192	288	384	576	768
Time per step	132.43s	69.79s	37.47s	27.12s	21.91s	17.45s	15.15s
Par. Efficiency	100.0	94.88	88.36	81.40	75.56	63.24	54.63
LBM Update	5.91	5.61	5.38	4.92	4.50	3.73	3.19
Regridding	15.44	12.02	11.38	10.92	10.02	8.94	8.24
Partitioning	4.16	2.43	1.16	1.02	1.04	1.16	1.34
Interpolation	3.76	3.53	3.33	3.05	2.83	2.37	2.06
Sync Boundaries	54.71	59.35	59.73	56.95	54.54	52.01	51.19
Sync Fixup	9.10	10.41	12.25	16.62	20.77	26.17	28.87
Level set	0.78	0.93	1.21	1.37	1.45	1.48	1.47
Interp./Extrap.	3.87	3.81	3.76	3.49	3.26	2.75	2.39
Misc	2.27	1.91	1.79	1.67	1.58	1.38	1.25

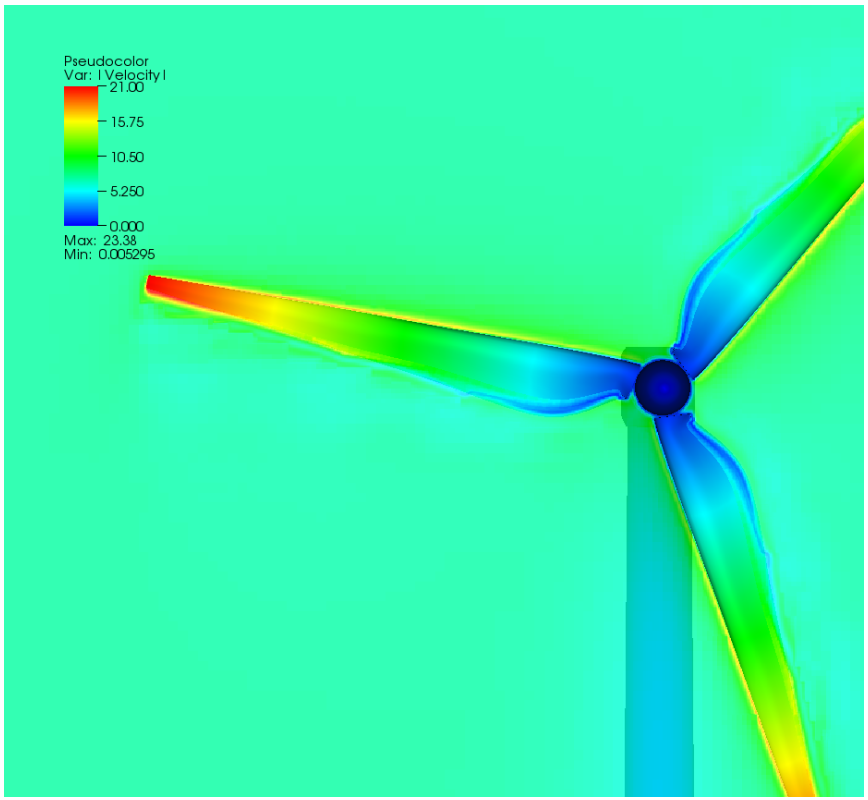
Simulation of a single turbine

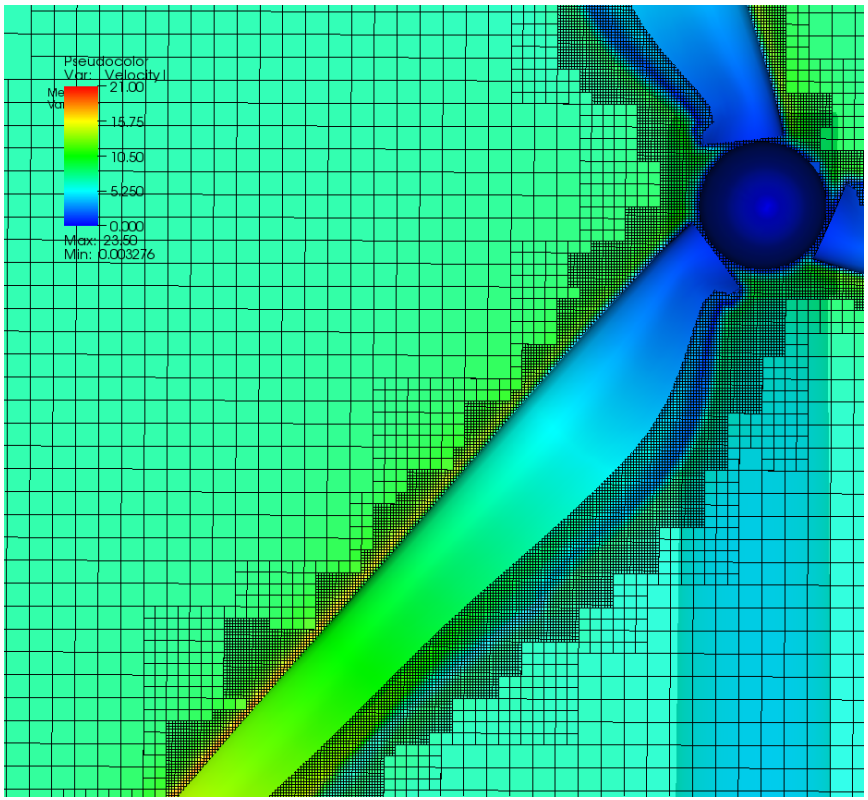
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.



Simulation of a single turbine

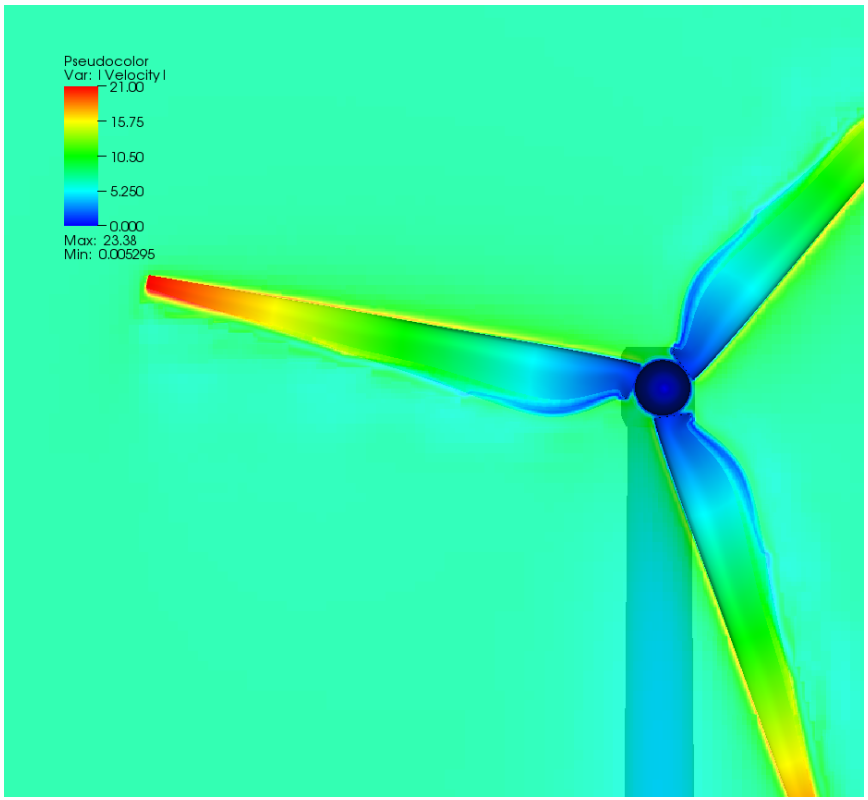
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.

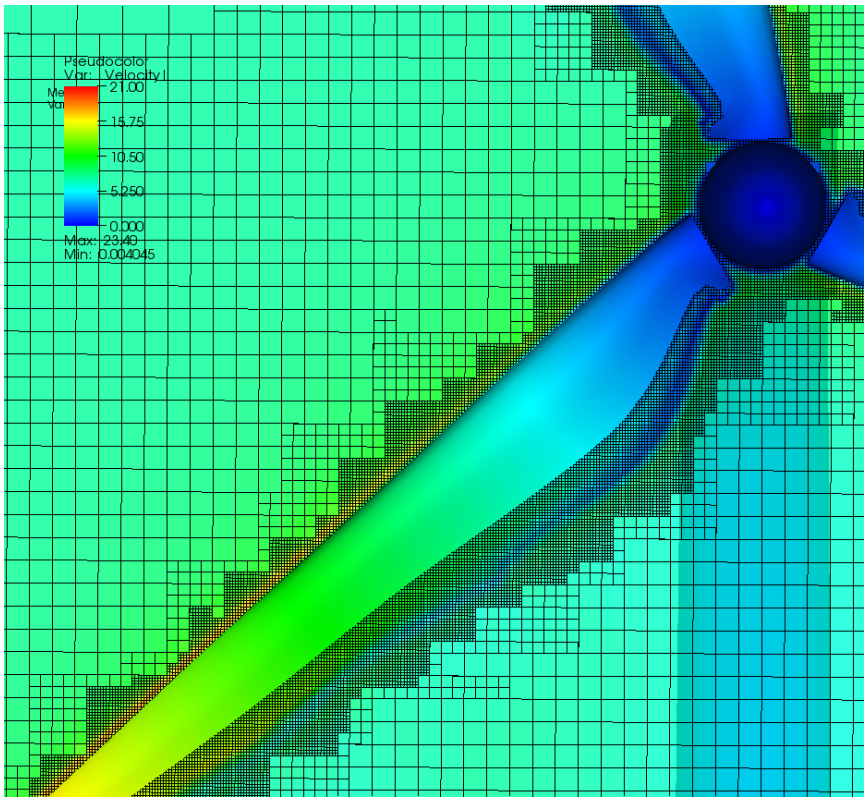




Simulation of a single turbine

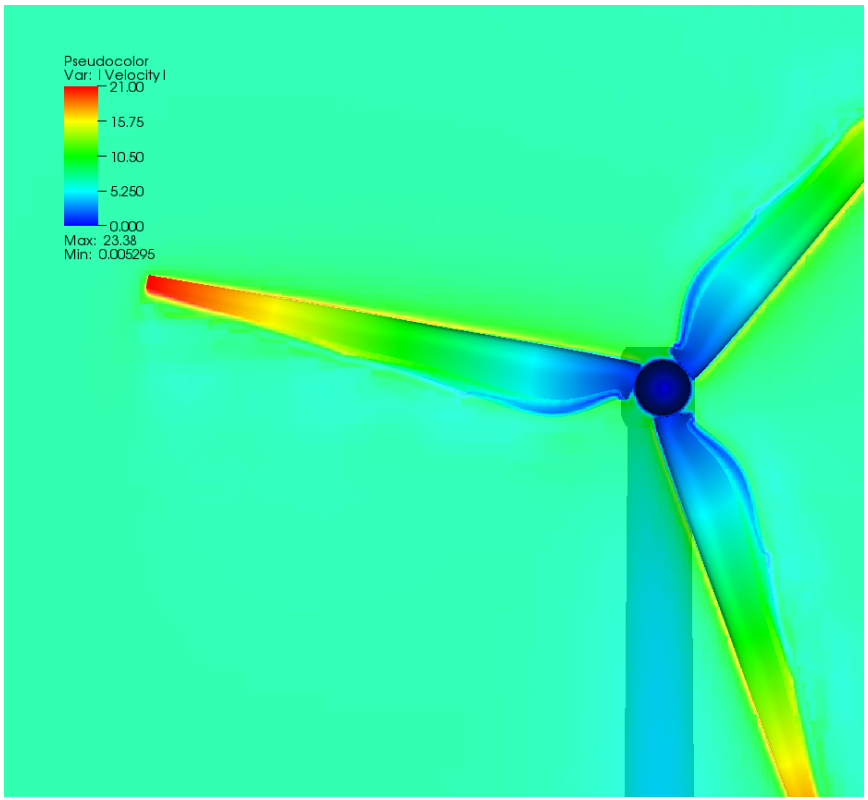
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.

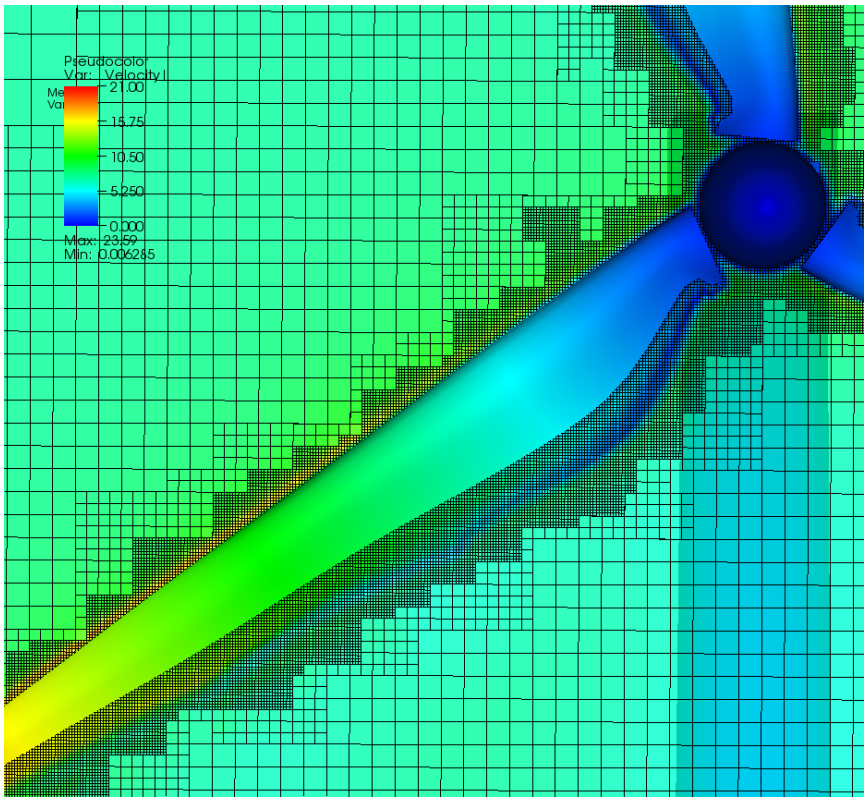




Simulation of a single turbine

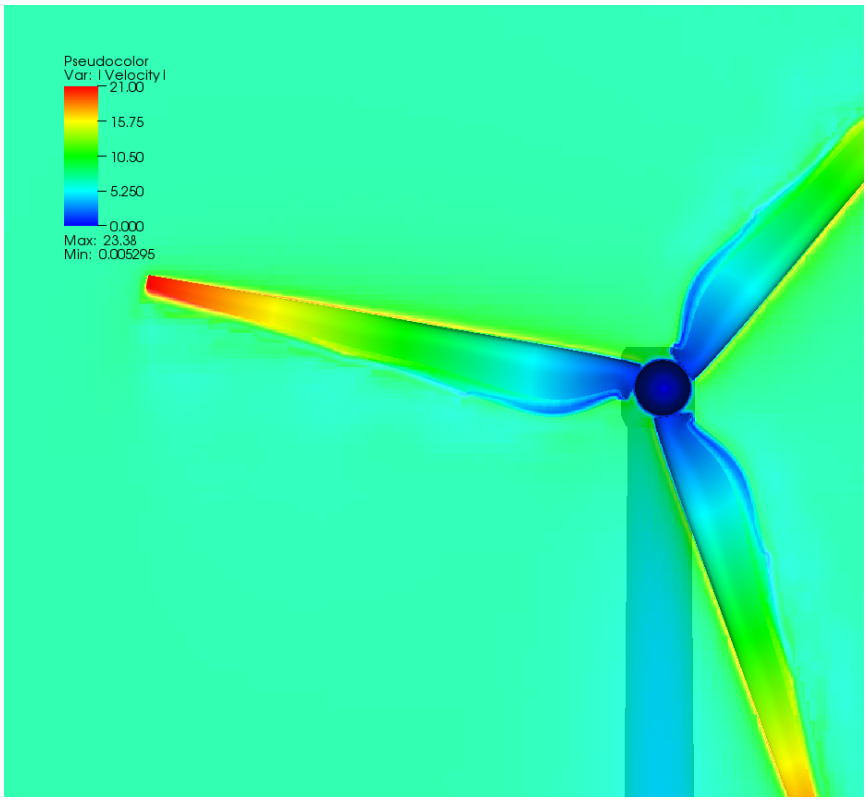
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.

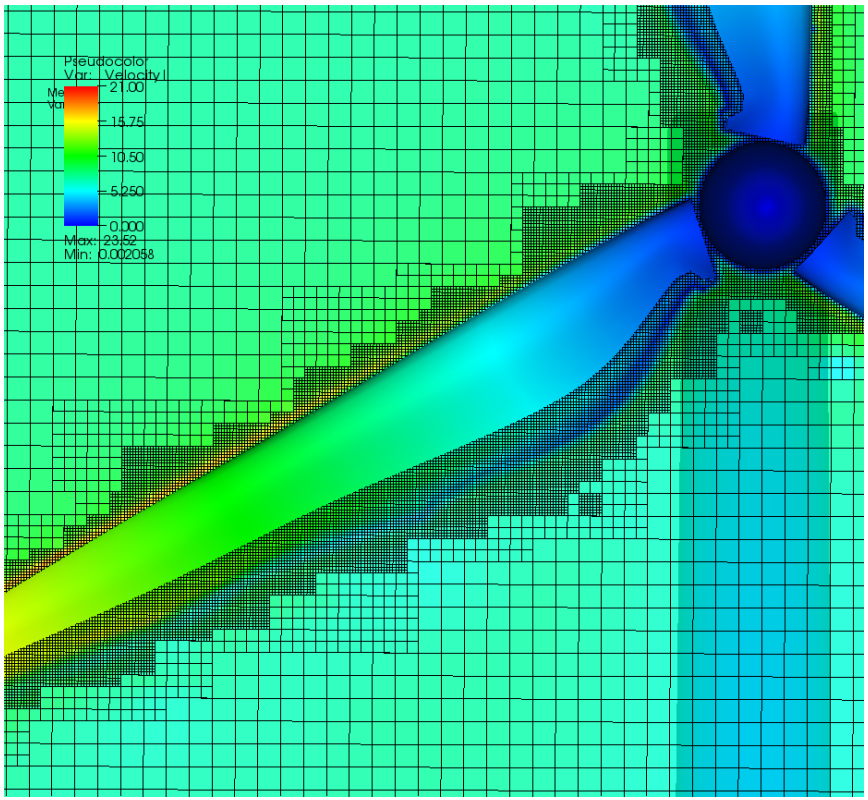




Simulation of a single turbine

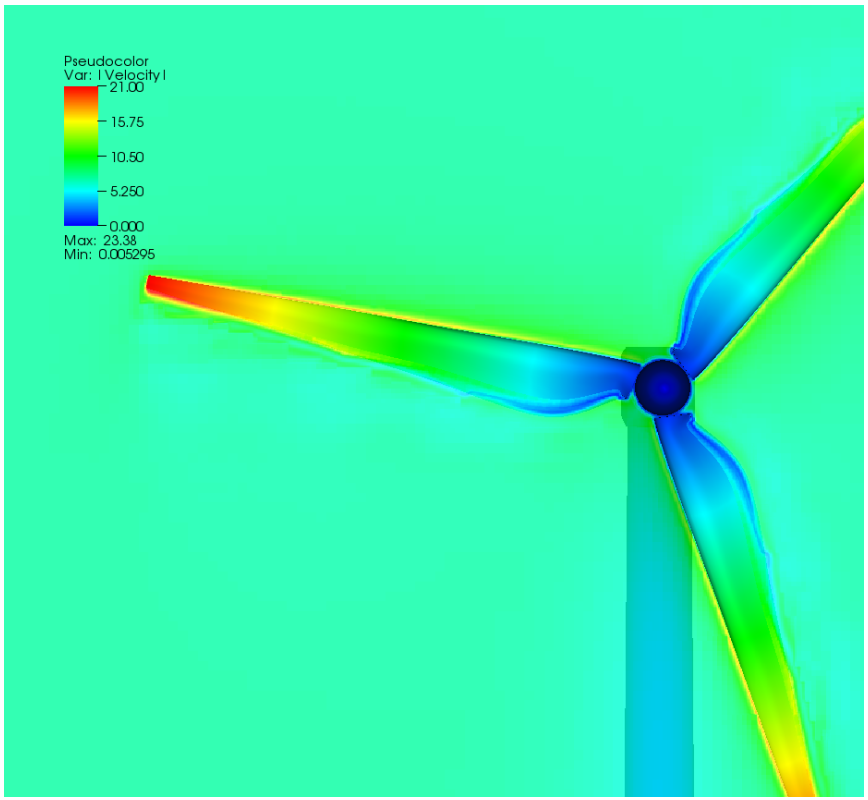
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.

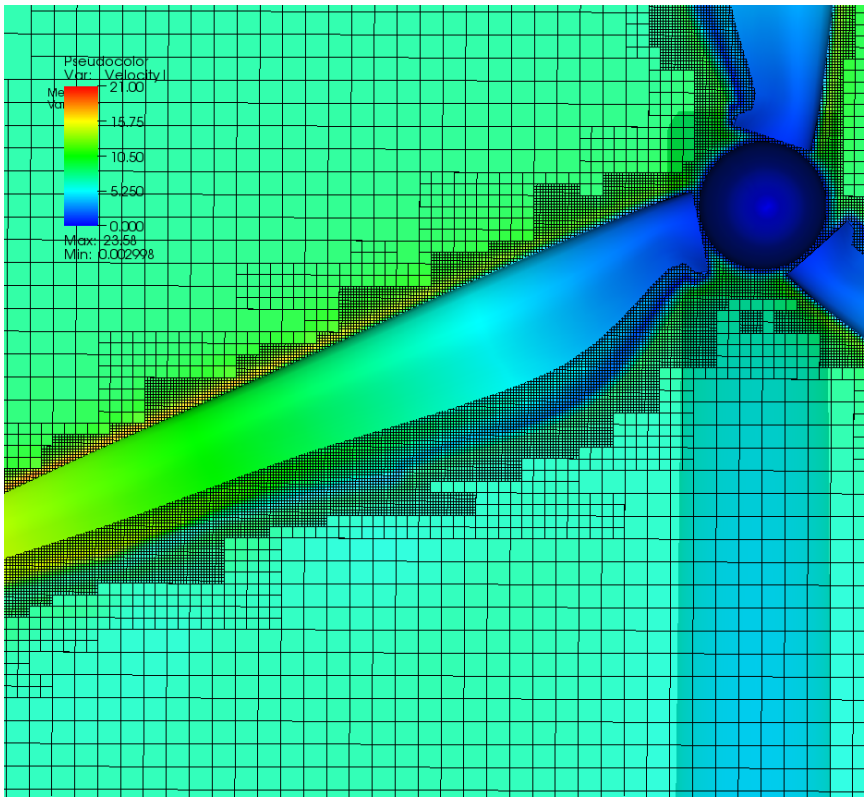




Simulation of a single turbine

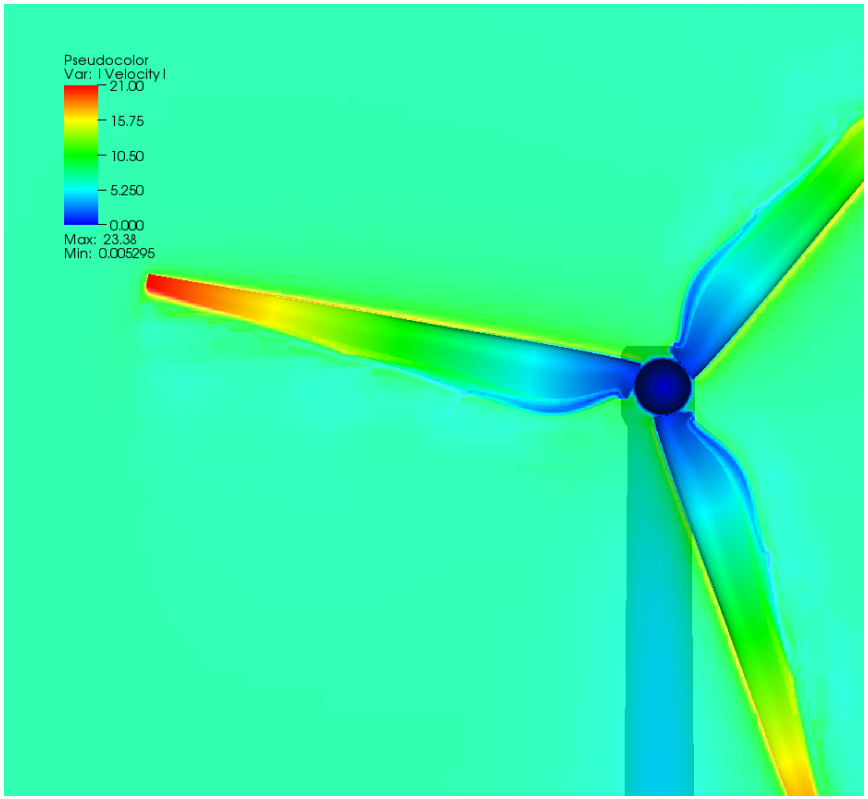
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.

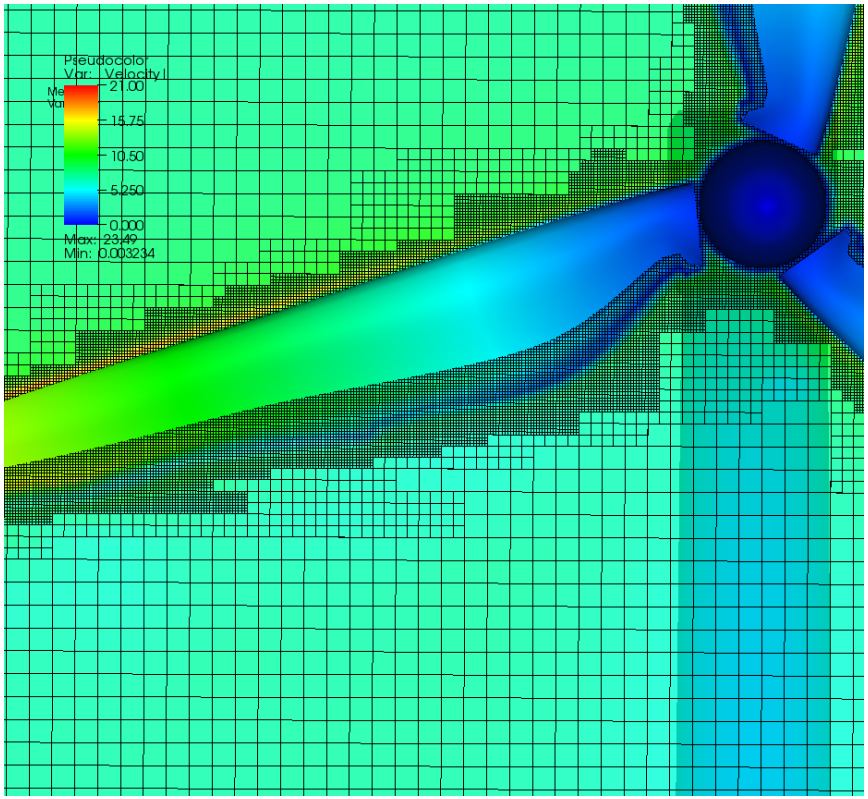




Simulation of a single turbine

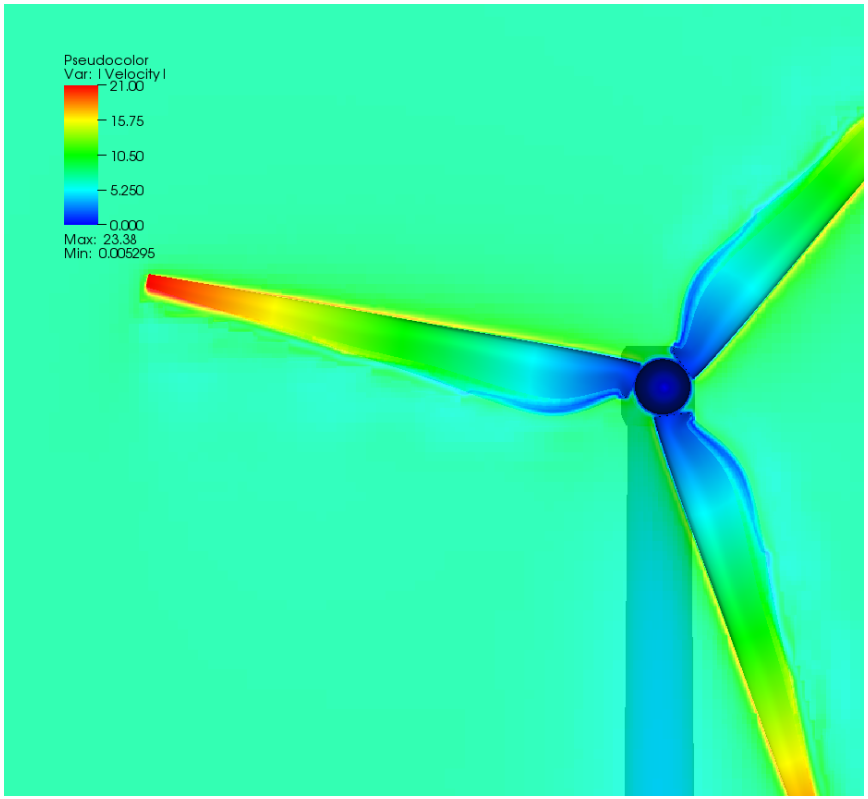
- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.

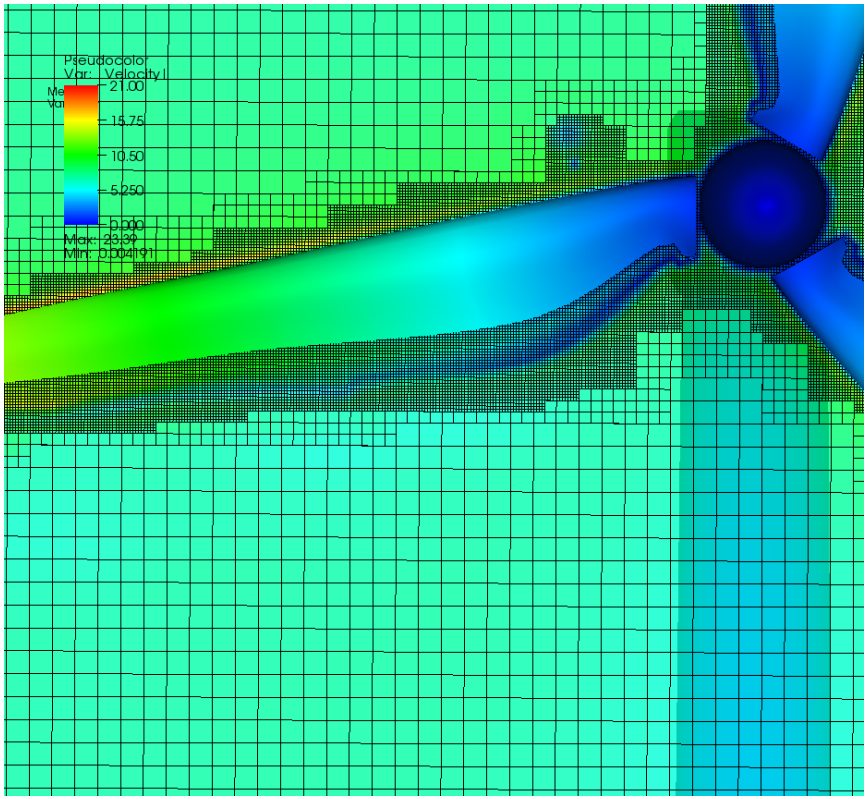




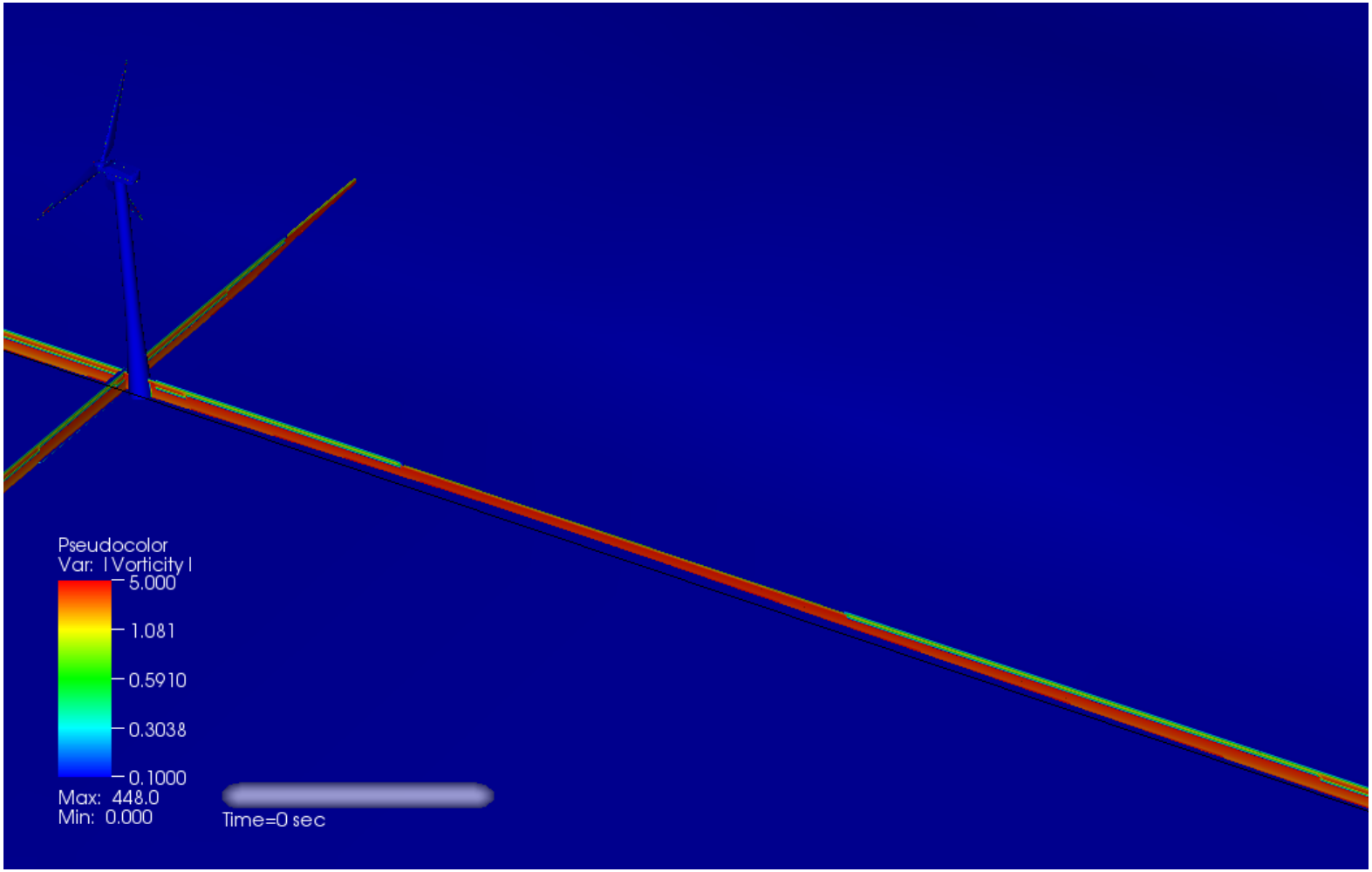
Simulation of a single turbine

- Geometry from realistic Vestas V27 turbine. Rotor diameter 27 m, tower height $\sim 35\,\mathrm{m}$. Ground considered. Structural model built by S. Wood (University of Tennessee, Knoxville).
- Prescribed motion of rotor with 15 rpm. Inflow velocity 7 $\mathrm{m/s}$.
- Simulation domain $200\,\mathrm{m} \times 100\,\mathrm{m} \times 100\,\mathrm{m}$.
- Base mesh $400 \times 200 \times 200$ cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 3.125\,\mathrm{cm}$.
- 141,344 highest level iterations to $t_e = 30\,\mathrm{s}$ computed.



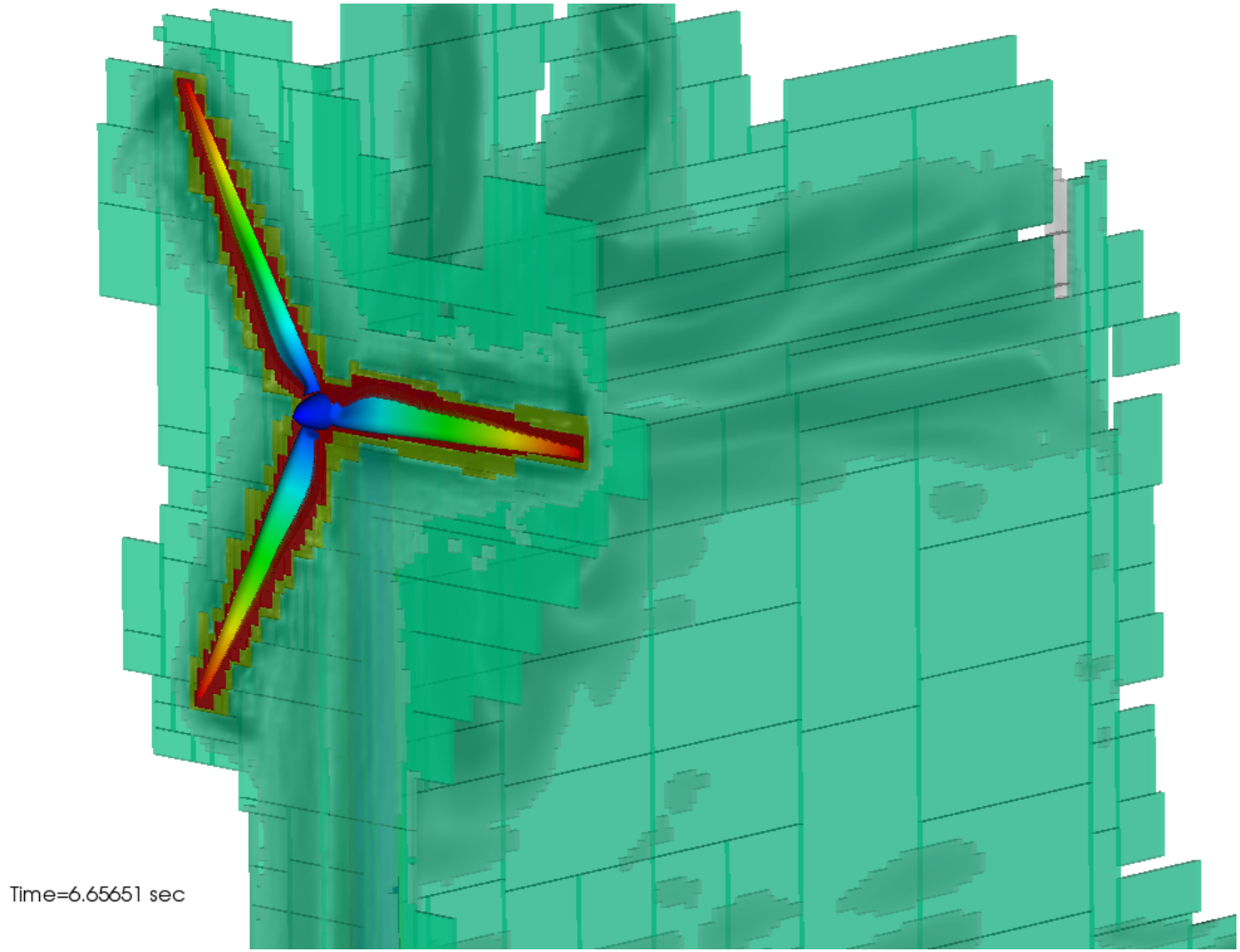


Wake field behind turbine



- Simulation on 96 cores Intel Xeon-Westmere. $\sim 10,400\,\mathrm{h}$ CPU.
- Error estimation in $|\mathbf{u}|$ refines wake up to level 1 ($\Delta x = 25\,\mathrm{cm}$).
- Rotation starts at $t = 4\,\mathrm{s}$.

Adaptive refinement

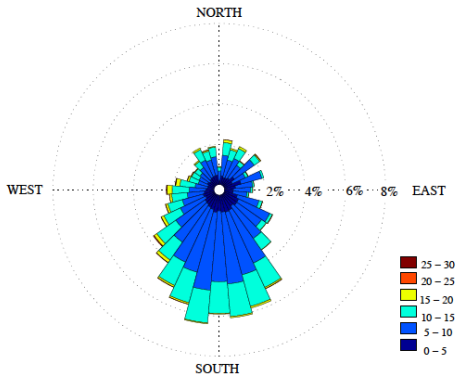


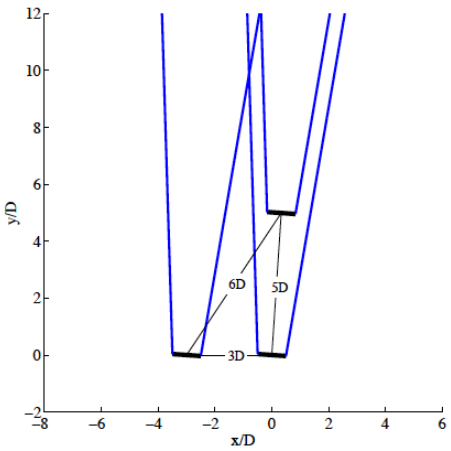
Dynamic evolution of refinement blocks (indicated by color).

Simulation of the SWIFT array

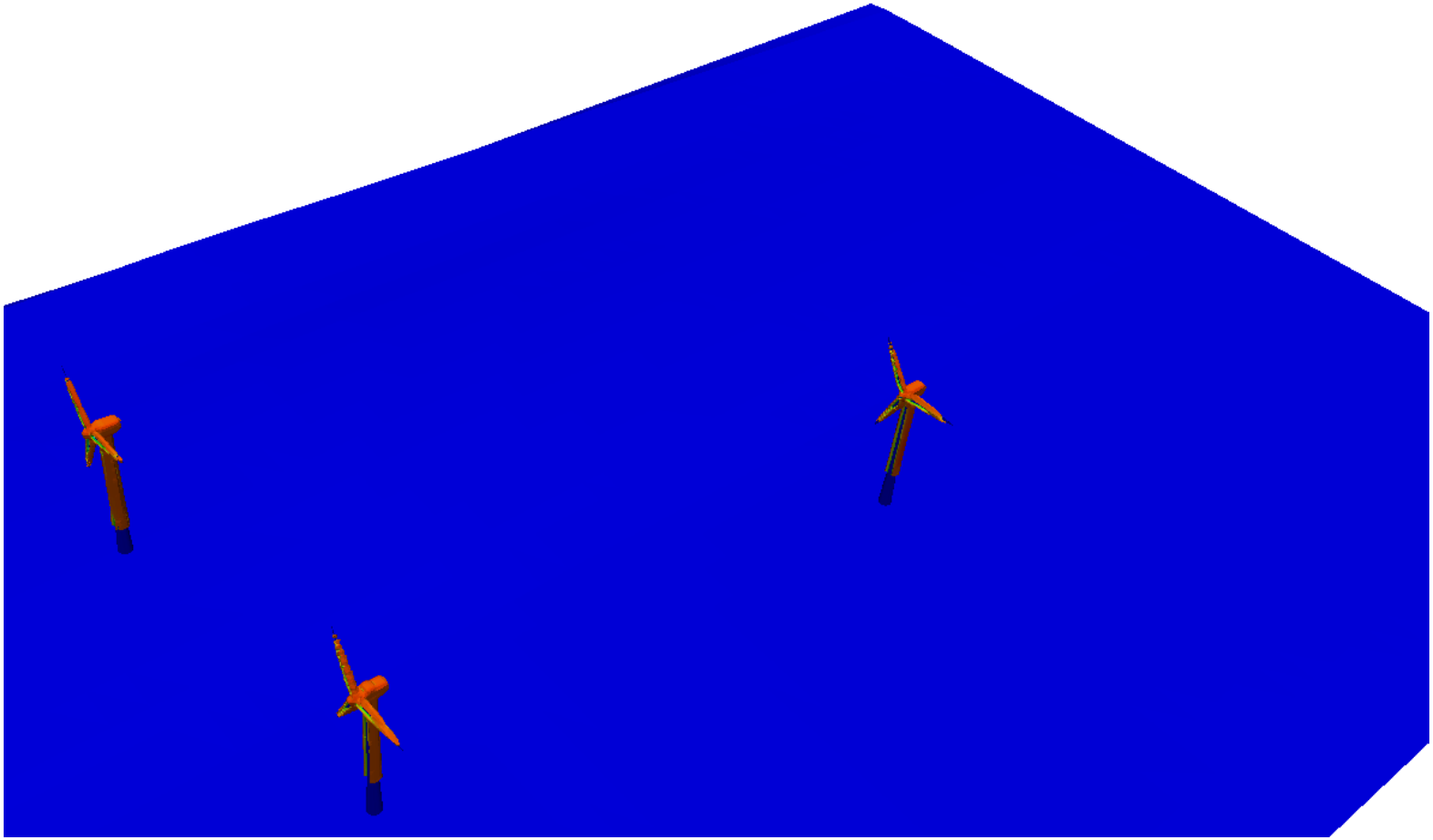
- Three Vestas V27 turbines. 225 kW power generation at wind speeds 14 to 25 m/s (then cut-off).
- Prescribed motion of rotor with 15 rpm and 43 rpm. Inflow velocity 7 m/s (power generation 52.5 kW) and 25 m/s.

- Simulation domain 488 m × 240 m × 100 m.
- Base mesh 448 × 240 × 100 cells with refinement factors 2,2,4. Resolution of rotor and tower $\Delta x = 6.25\,\mathrm{cm}$.
- 94,224 highest level iterations to $t_e = 40\,\mathrm{s}$ computed.



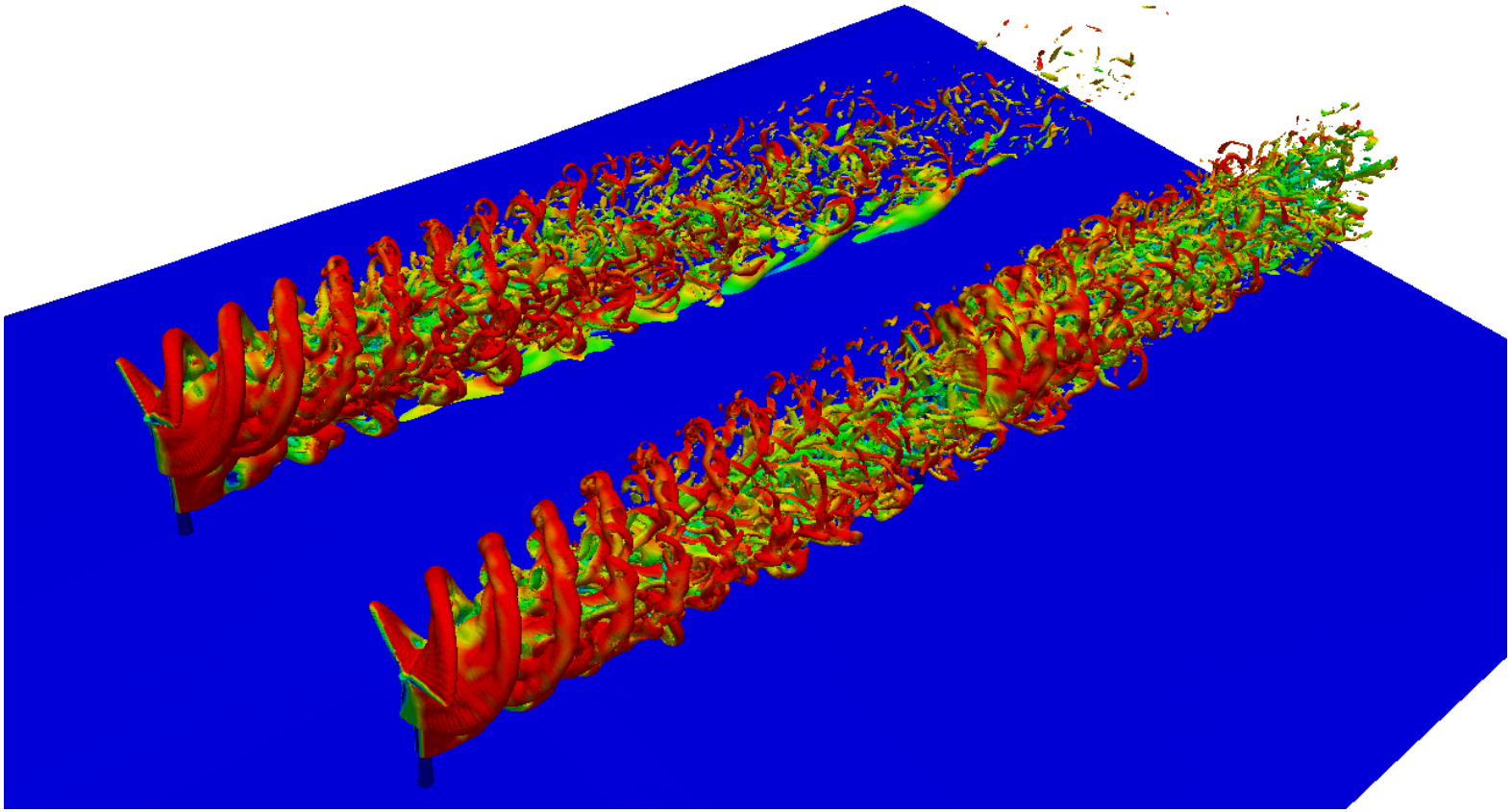


Wake build-up – 25 m/s, 43 rpm



- On 288 cores Intel Ivybride 10 s in 38.5 h (11,090 h CPU)
- Only levels 0 and 1 used for iso-surface visualization

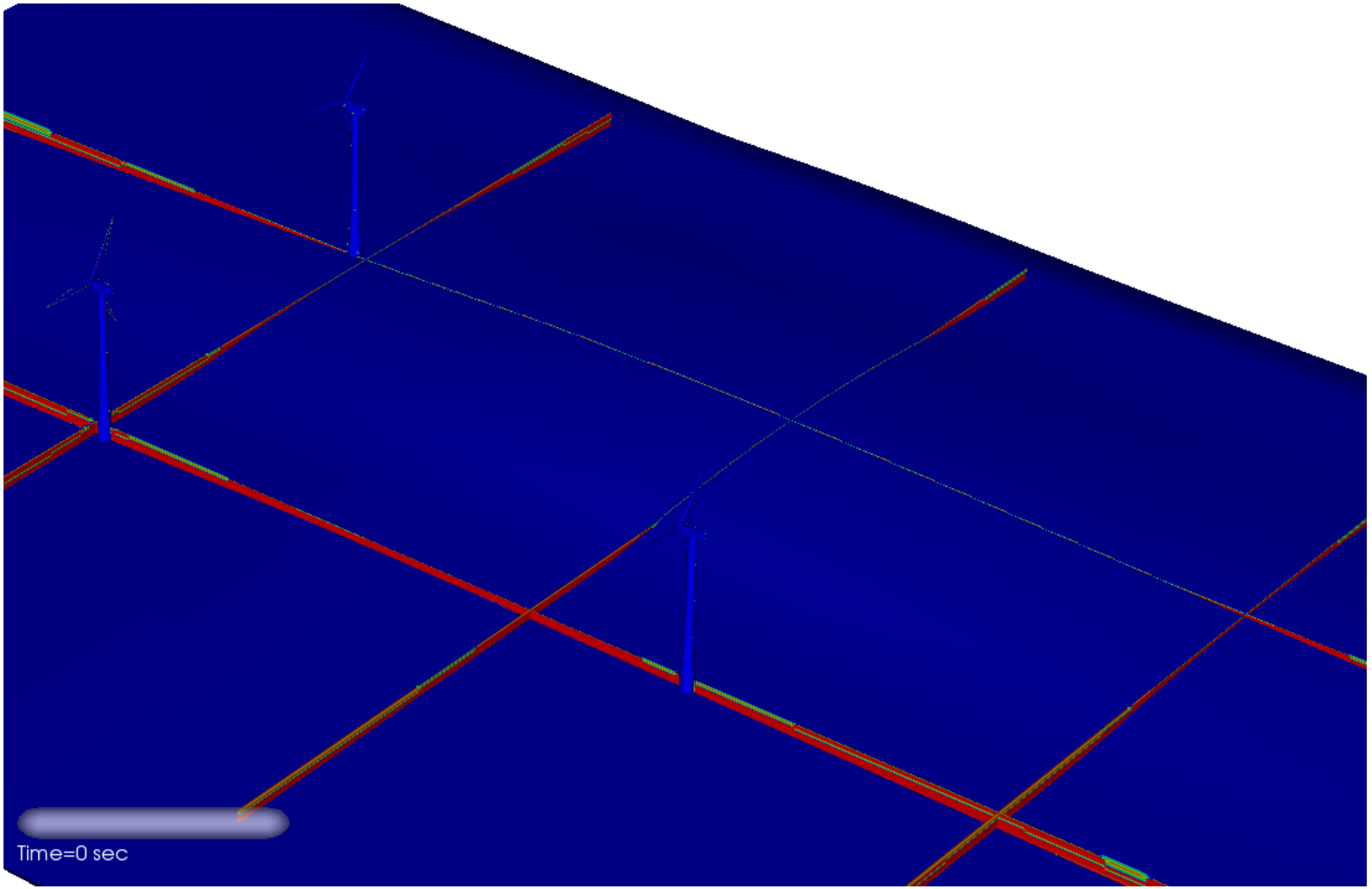
Wake propagation through array – 25 m/s, 43 rpm



- At t_e approximately 140M cells used vs. 44 billion (factor 315)
- Only levels 0 and 1 used for iso-surface visualization

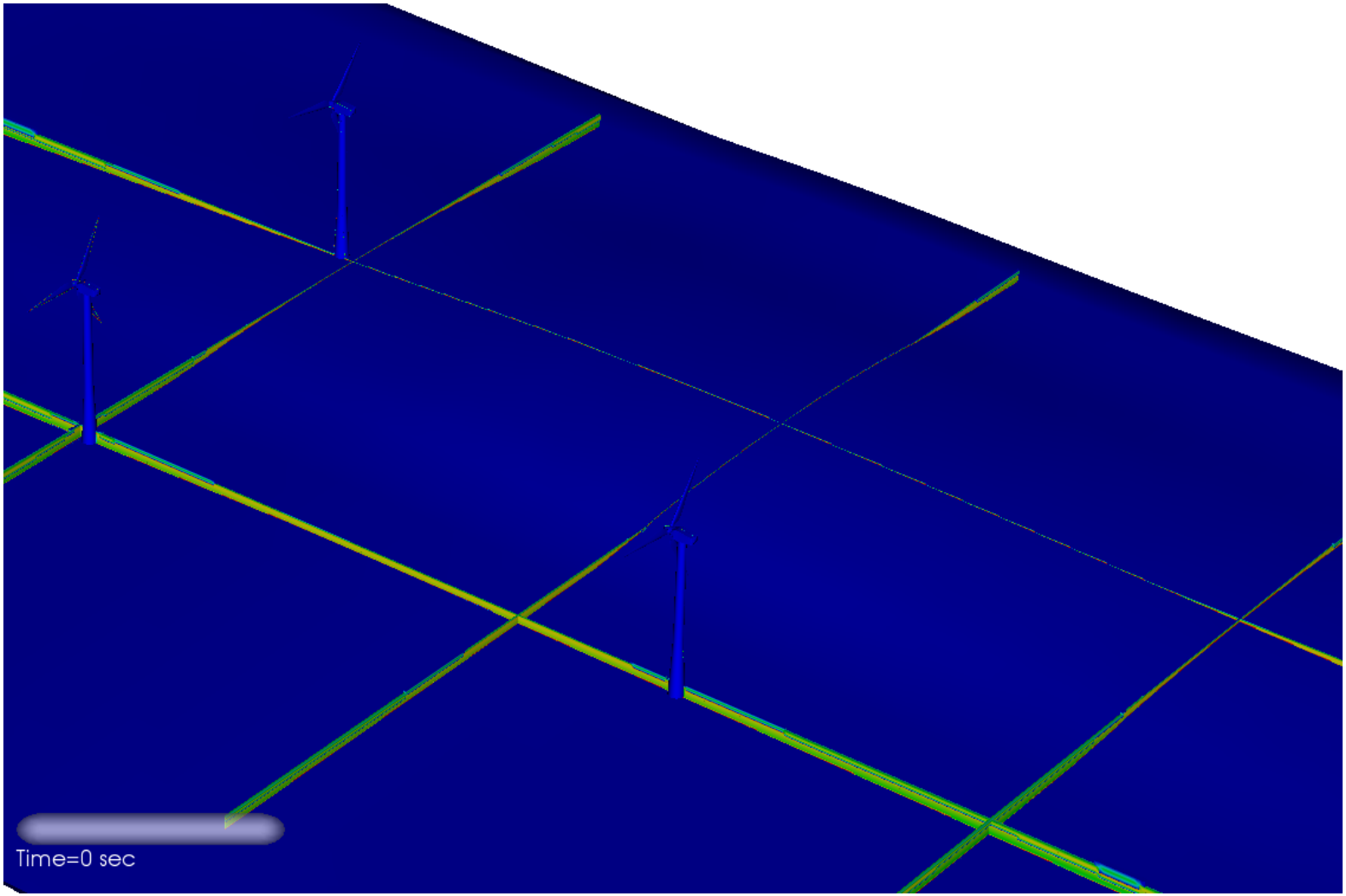
Level	Grids	Cells
0	3,234	10,752,000
1	11,921	21,020,256
2	66,974	102,918,568
3	896	5,116,992

Vorticity generation – 25 m/s, 43 rpm



- Refinement of wake up to level 2 ($\Delta x = 25\,\mathrm{cm}$).
- Rotation starts at $t = 4\,\mathrm{s}$, full refinement at $t = 8\,\mathrm{s}$ to avoid refining initial acoustic waves.

Vorticity generation – 7 m/s, 15 rpm



- Number of wakes between turbines increased from $\sim$ 12 to $\sim$ 15 but vorticity production visibly reduced

Conclusions

- Developed and demonstrated a first version of a block-based dynamically adaptive LBM for real-world CFD with moving boundaries.
- Reuse of templatized AMROC classes from previous finite volume methods already provides robust real-world capabilities.

Conclusions

- Developed and demonstrated a first version of a block-based dynamically adaptive LBM for real-world CFD with moving boundaries.
- Reuse of templatized AMROC classes from previous finite volume methods already provides robust real-world capabilities.
- Currently developing dynamic Smagorinsky turbulence model and turbulent validation cases, especially for airfoils and rotating geometry
- Need to develop wall-function model for high Re flows

Conclusions

- Developed and demonstrated a first version of a block-based dynamically adaptive LBM for real-world CFD with moving boundaries.
- Reuse of templatized AMROC classes from previous finite volume methods already provides robust real-world capabilities.
- Currently developing dynamic Smagorinsky turbulence model and turbulent validation cases, especially for airfoils and rotating geometry
- Need to develop wall-function model for high Re flows
- Some more developments for post-processing (friction coefficients), time-averaging of fields

Conclusions

- Developed and demonstrated a first version of a block-based dynamically adaptive LBM for real-world CFD with moving boundaries.
- Reuse of templatized AMROC classes from previous finite volume methods already provides robust real-world capabilities.
- Currently developing dynamic Smagorinsky turbulence model and turbulent validation cases, especially for airfoils and rotating geometry
- Need to develop wall-function model for high Re flows
- Some more developments for post-processing (friction coefficients), time-averaging of fields
- Realistic wind turbine model with dynamic pitch angle, nacelle rotation, etc. under development (S. Wood, UT Knoxville), idea is to incorporate FAST by NREL

Conclusions

- Developed and demonstrated a first version of a block-based dynamically adaptive LBM for real-world CFD with moving boundaries.
- Reuse of templatized AMROC classes from previous finite volume methods already provides robust real-world capabilities.
- Currently developing dynamic Smagorinsky turbulence model and turbulent validation cases, especially for airfoils and rotating geometry
- Need to develop wall-function model for high Re flows
- Some more developments for post-processing (friction coefficients), time-averaging of fields
- Realistic wind turbine model with dynamic pitch angle, nacelle rotation, etc. under development (S. Wood, UT Knoxville), idea is to incorporate FAST by NREL
- Currently also implementing LBM in 3d with Boussinesq approximation for thermal convection and fully compressible LBM
- Performance for moderate core count is reasonable, some improvements for larger core count still desirable. Looking into GPU and Phi coprocessor versions.

References I

[Bell et al., 1994] Bell, J., Berger, M., Saltzman, J., and Welcome, M. (1994). Three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Comp.*, 15(1):127–138.

[Berger, 1986] Berger, M. (1986). Data structures for adaptive grid generation. *SIAM J. Sci. Stat. Comput.*, 7(3):904–916.

[Berger and Colella, 1988] Berger, M. and Colella, P. (1988). Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:64–84.

[Berger and Oliger, 1984] Berger, M. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512.

[Berger and Rigoutsos, 1991] Berger, M. and Rigoutsos, I. (1991). An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1278–1286.

[Chen et al., 2006] Chen, H., Filippova, O., Hoch, J., Molvig, K., Shock, R., Teixeira, C., and Zhang, R. (2006). Grid refinement in lattice Boltzmann methods based on volumetric formulation. *Physica A*, 362:158–167.

References II

[Deiterding, 2011] Deiterding, R. (2011). Block-structured adaptive mesh refinement - theory, implementation and application. *European Series in Applied and Industrial Mathematics: Proceedings*, 34:97–150.

[Deiterding et al., 2009] Deiterding, R., Cirak, F., and Mauch, S. P. (2009). Efficient fluid-structure interaction simulation of viscoplastic and fracturing thin-shells subjected to underwater shock loading. In Hartmann, S., Meister, A., Schäfer, M., and Turek, S., editors, *Int. Workshop on Fluid-Structure Interaction. Theory, Numerics and Applications, Herrsching am Ammersee 2008*, pages 65–80. kassel university press GmbH.

[Deiterding et al., 2007] Deiterding, R., Cirak, F., Mauch, S. P., and Meiron, D. I. (2007). A virtual test facility for simulating detonation- and shock-induced deformation and fracture of thin flexible shells. *Int. J. Multiscale Computational Engineering*, 5(1):47–63.

[Deiterding et al., 2006] Deiterding, R., Radovitzky, R., Mauch, S. P., Noels, L., Cummings, J. C., and Meiron, D. I. (2006). A virtual test facility for the efficient simulation of solid materials under high energy shock-wave loading. *Engineering with Computers*, 22(3-4):325–347.

References III

[Deiterding and Wood, 2013] Deiterding, R. and Wood, S. L. (2013). Parallel adaptive fluid-structure interaction simulations of explosions impacting building structures. *Computers & Fluids*, 88:719–729.

[Hähnel, 2004] Hähnel, D., editor (2004). *Molekulare Gasdynamik*. Springer.

[Henderson, 1995] Henderson, R. D. (1995). Details of the drag curve near the onset of vortex shedding. *Phys. Fluids*, 7:2102–2104.

[Hou et al., 1996] Hou, S., Sterling, J., Chen, S., and Doolen, G. D. (1996). A lattice Boltzmann subgrid model for high Reynolds number flows. In Lawniczak, A. T. and Kapral, R., editors, *Pattern formation and lattice gas automata*, volume 6, pages 151–166. Fields Inst Comm.

[Mauch, 2000] Mauch, S. (2000). A fast algorithm for computing the closest point and distance transform. *SIAM J. Scientific Comput.*

[Mauch, 2003] Mauch, S. P. (2003). *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, California Institute of Technology.

References IV

[Peng and Luo, 2008] Peng, Y. and Luo, L.-S. (2008). A comparative study of immersed-boundary and interpolated bounce-back methods in lbe. *Prog. Comp. Fluid Dynamics*, 8(1-4):156–167.

[Schlaffer, 2013] Schlaffer, M. B. (2013). *Non-reflecting boundary conditions for the lattice Boltzmann method*. PhD thesis, Technical University Munich.

[Sethian, 1999] Sethian, J. A. (1999). *Level set methods and fast marching methods*. Cambridge University Press, Cambridge, New York.

[Yu, 2004] Yu, H. (2004). *Lattice Boltzmann equation simulations of turbulence, mixing, and combustion*. PhD thesis, Texas A&M University.

The basic recursive algorithm

```
AdvanceLevel(l)

  Repeat r_l times
      Set ghost cells of Q^l(t)


      UpdateLevel(l)




      t := t + Δt_l
```

The basic recursive algorithm

```
AdvanceLevel(l)

  Repeat r_l times
      Set ghost cells of Q^l(t)


      UpdateLevel(l)
      If level l+1 exists?
          Set ghost cells of Q^l(t + Δt_l)
          AdvanceLevel(l+1)


      t := t + Δt_l
```

- Recursion

The basic recursive algorithm

```
AdvanceLevel(l)

  Repeat r_l times
      Set ghost cells of Q^l(t)


      UpdateLevel(l)
      If level l+1 exists?
          Set ghost cells of Q^l(t + Δt_l)
          AdvanceLevel(l + 1)
          Average Q^{l+1}(t + Δt_l) onto Q^l(t + Δt_l)
          Correct Q^l(t + Δt_l) with δF^{l+1}
      t := t + Δt_l
```

- Recursion
- Restriction and flux correction

The basic recursive algorithm

```
AdvanceLevel(l)

  Repeat r_l times
      Set ghost cells of Q^l(t)
      If time to regrid?
          Regrid(l)
      UpdateLevel(l)
      If level l+1 exists?
          Set ghost cells of Q^l(t + Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t + Δt_l) onto Q^l(t + Δt_l)
          Correct Q^l(t + Δt_l) with δF^{l+1}
      t := t + Δt_l
```

- Recursion
- Restriction and flux correction
- Re-organization of hierarchical data

The basic recursive algorithm

```
AdvanceLevel(l)

  Repeat r_l times
      Set ghost cells of Q^l(t)
      If time to regrid?
          Regrid(l)
      UpdateLevel(l)
      If level l+1 exists?
          Set ghost cells of Q^l(t + Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t + Δt_l) onto Q^l(t + Δt_l)
          Correct Q^l(t + Δt_l) with δF^{l+1}
      t := t + Δt_l

Start - Start integration on level 0

    l = 0,  r_0 = 1
    AdvanceLevel(l)
```

- Recursion
- Restriction and flux correction
- Re-organization of hierarchical data

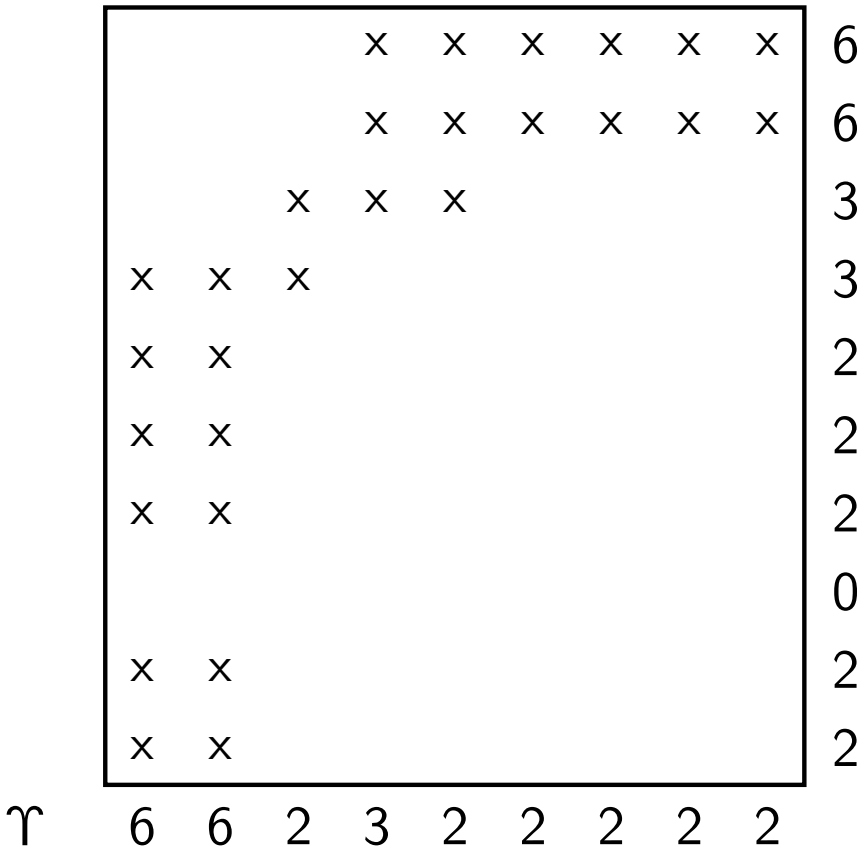
The basic recursive algorithm

```
AdvanceLevel(l)

  Repeat r_l times
      Set ghost cells of Q^l(t)
      If time to regrid?
          Regrid(l)
      UpdateLevel(l)
      If level l+1 exists?
          Set ghost cells of Q^l(t + Δt_l)
          AdvanceLevel(l+1)
          Average Q^{l+1}(t + Δt_l) onto Q^l(t + Δt_l)
          Correct Q^l(t + Δt_l) with δF^{l+1}
      t := t + Δt_l

Start - Start integration on level 0

    l = 0,  r_0 = 1
    AdvanceLevel(l)
```

- Recursion
- Restriction and flux correction
- Re-organization of hierarchical data

[Berger and Colella, 1988][Berger and Oliger, 1984]

Clustering by signatures

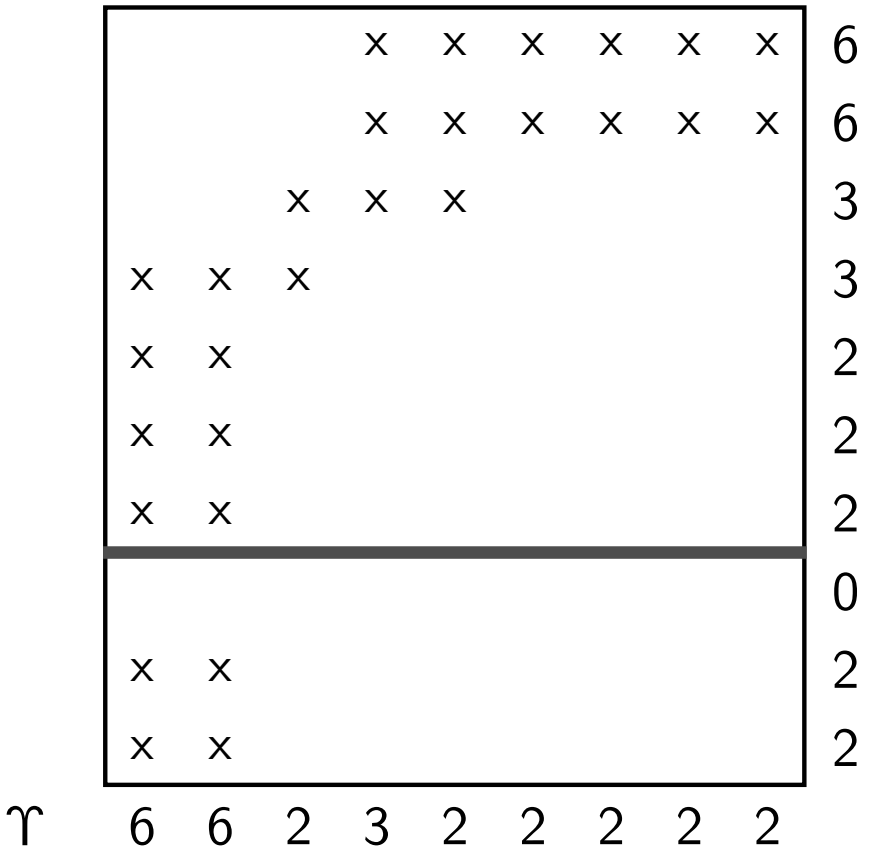


ϒ Flagged cells per row/column

Δ Second derivative of ϒ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_{\nu} + \Upsilon_{\nu-1}$

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]

Clustering by signatures

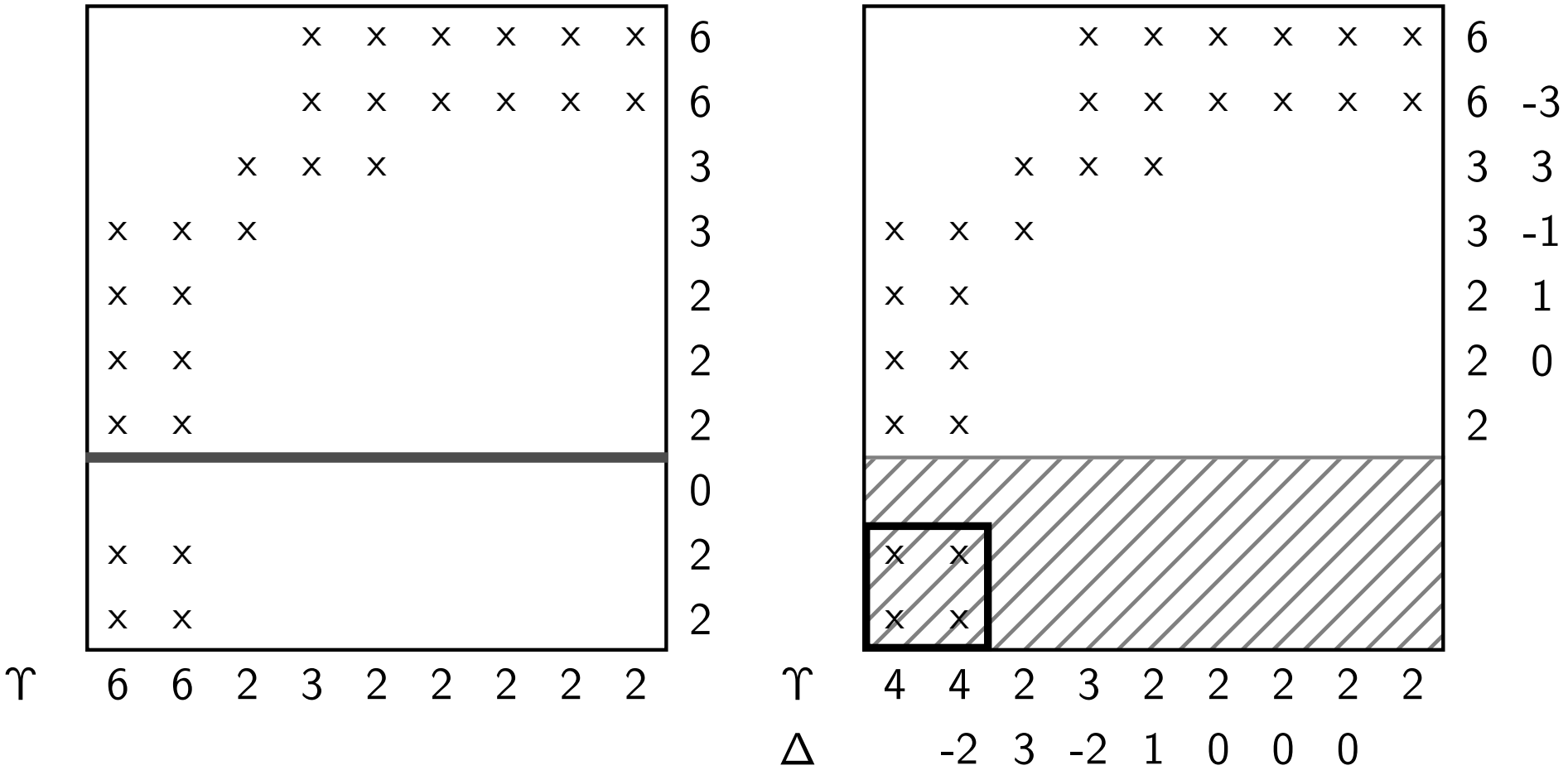


Υ Flagged cells per row/column

Δ Second derivative of Υ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_{\nu} + \Upsilon_{\nu-1}$

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]

Clustering by signatures

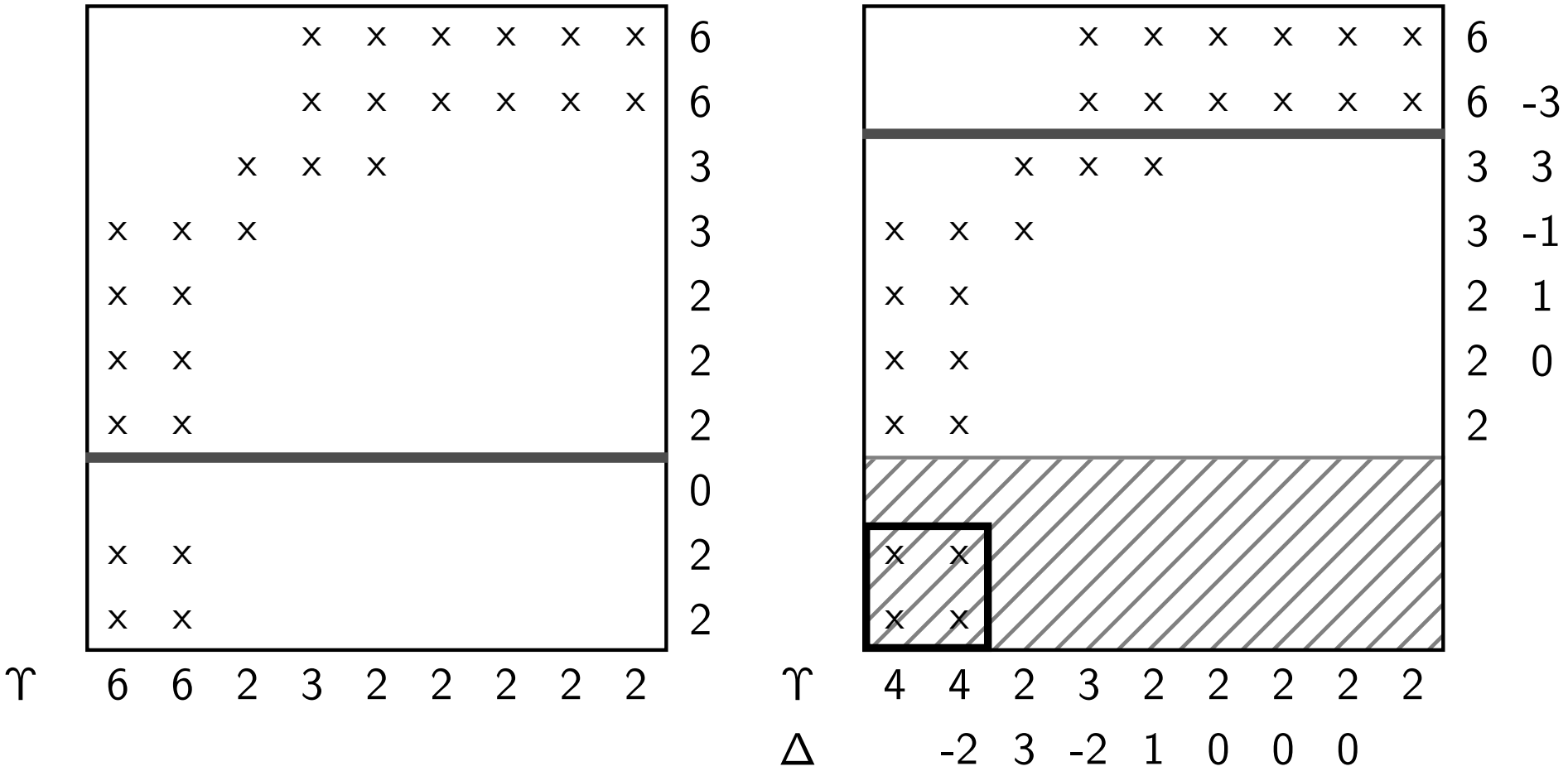


ϒ Flagged cells per row/column

Δ Second derivative of ϒ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_{\nu} + \Upsilon_{\nu-1}$

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]

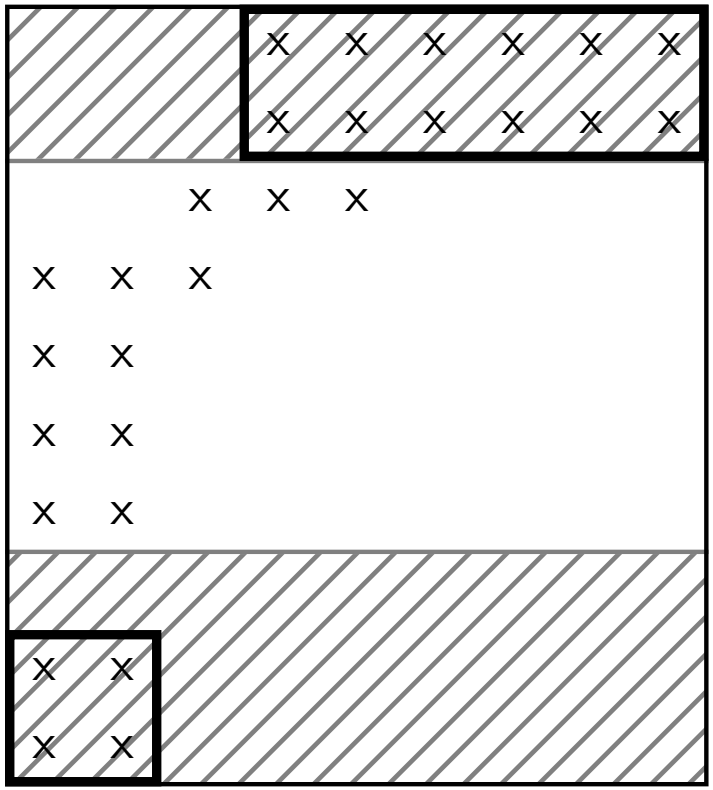
Clustering by signatures



ϒ Flagged cells per row/column

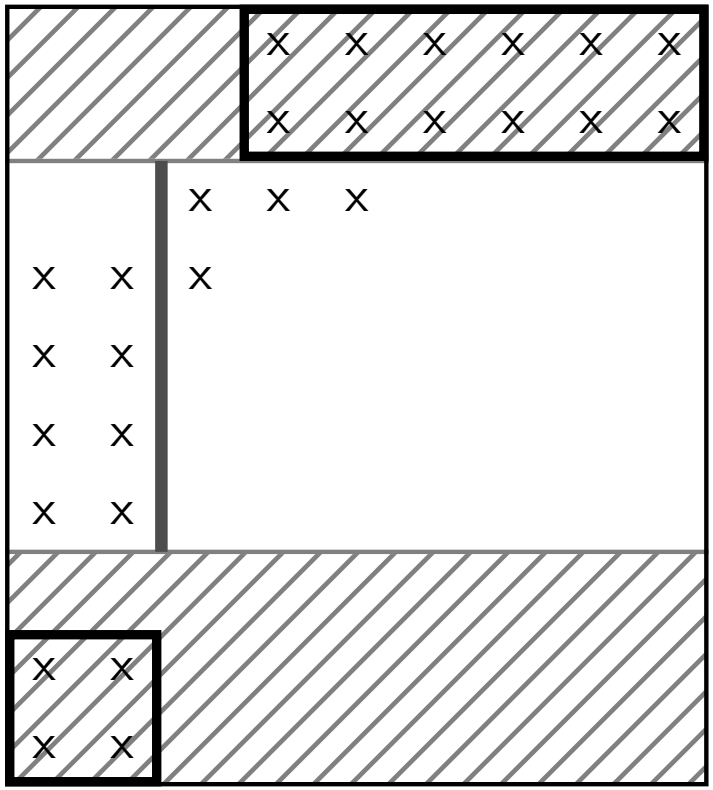
Δ Second derivative of ϒ, $\Delta = \Upsilon_{\nu+1} - 2\,\Upsilon_{\nu} + \Upsilon_{\nu-1}$

Technique from image detection: [Bell et al., 1994], see also [Berger and Rigoutsos, 1991], [Berger, 1986]



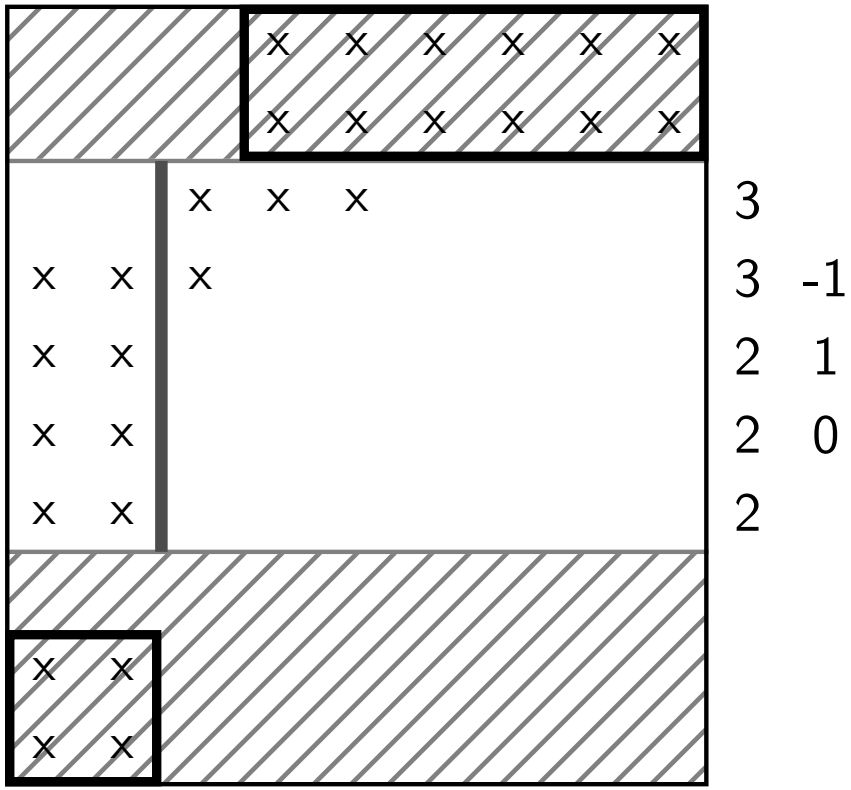
Recursive generation of $\breve{G}_{l,m}$

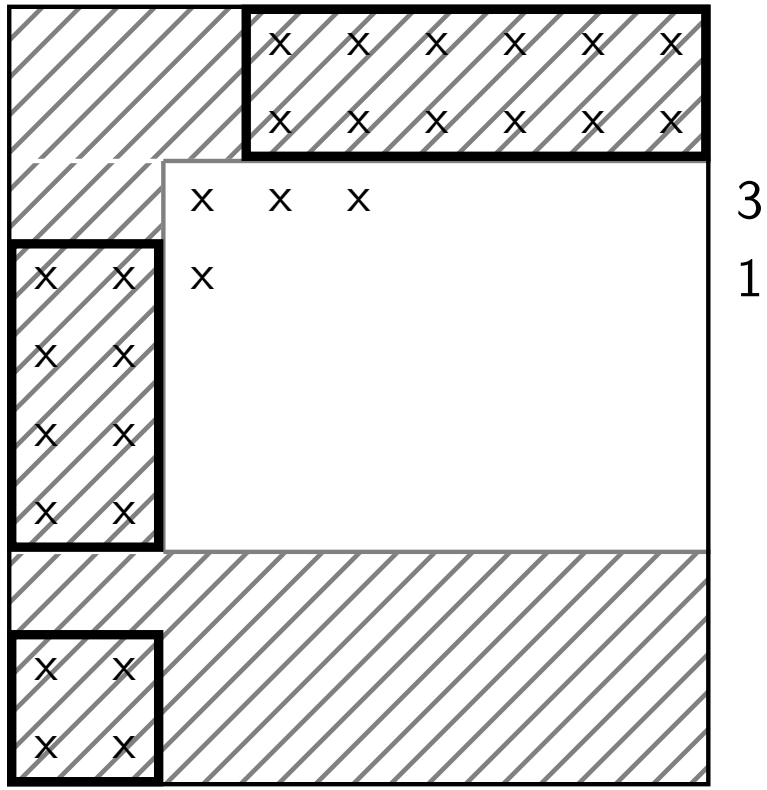
1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$



Recursive generation of $\breve{G}_{l,m}$

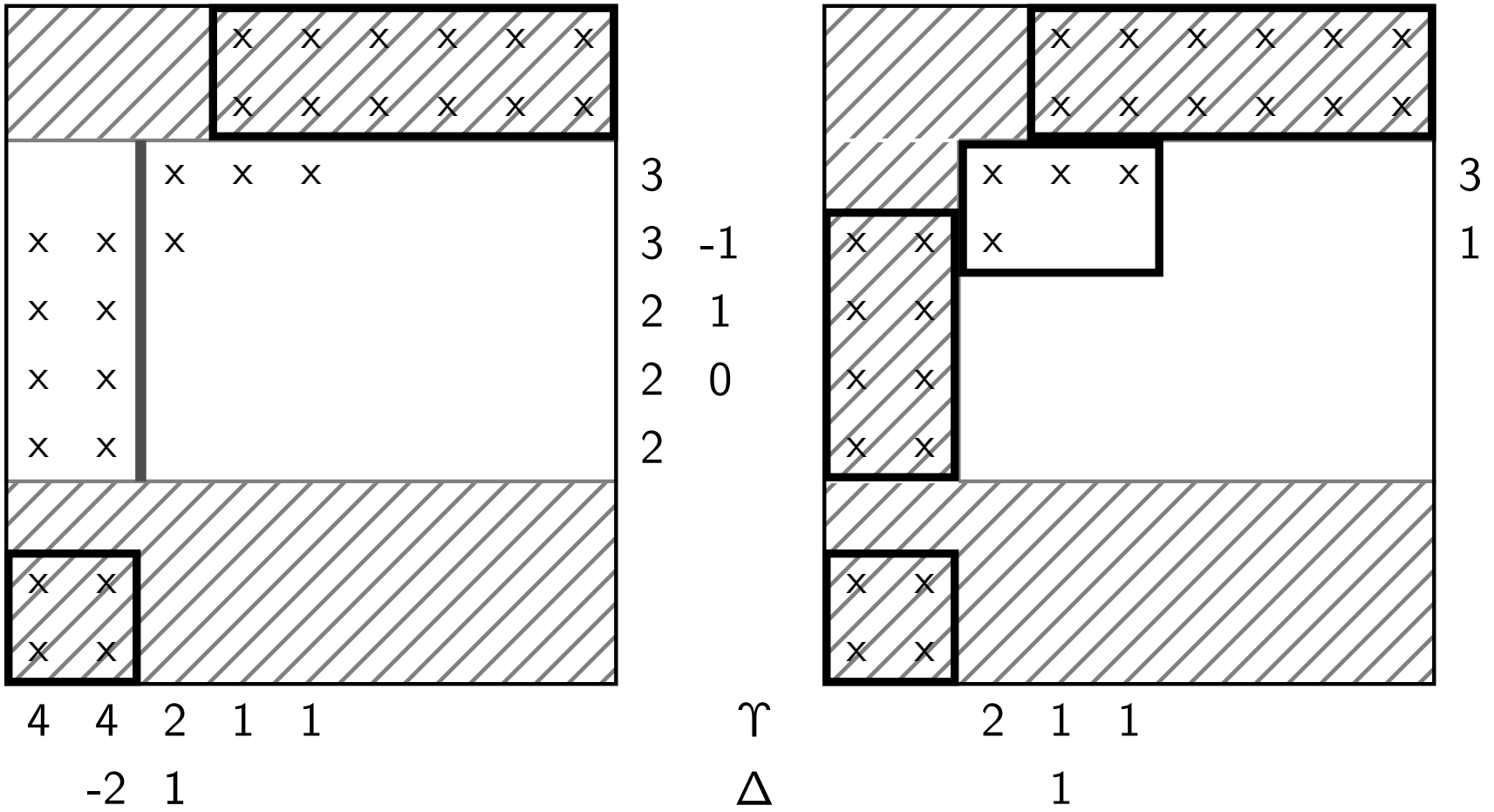
1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$





Recursive generation of $\breve{G}_{l,m}$

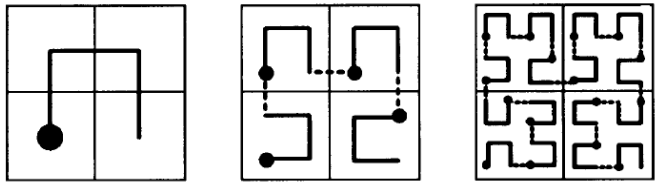
1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$

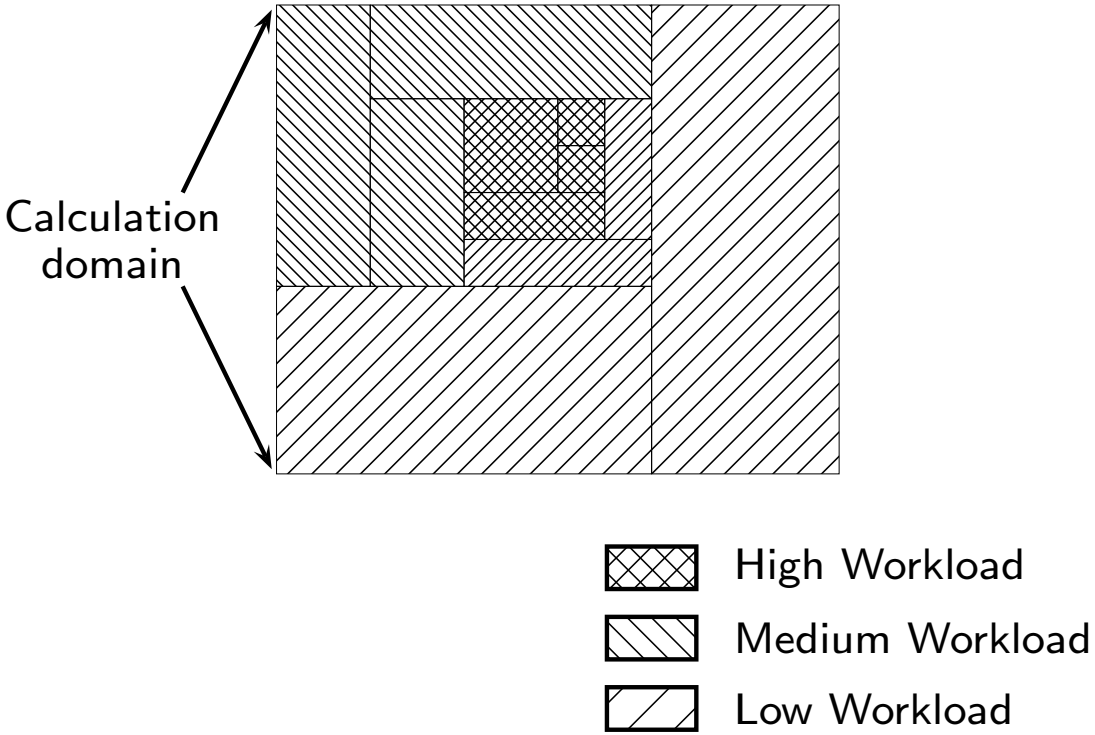


Recursive generation of $\breve{G}_{l,m}$

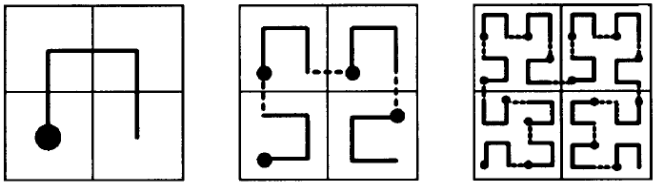
1. 0 in Υ
2. Largest difference in Δ
3. Stop if ratio between flagged and unflagged cell $> \eta_{tol}$

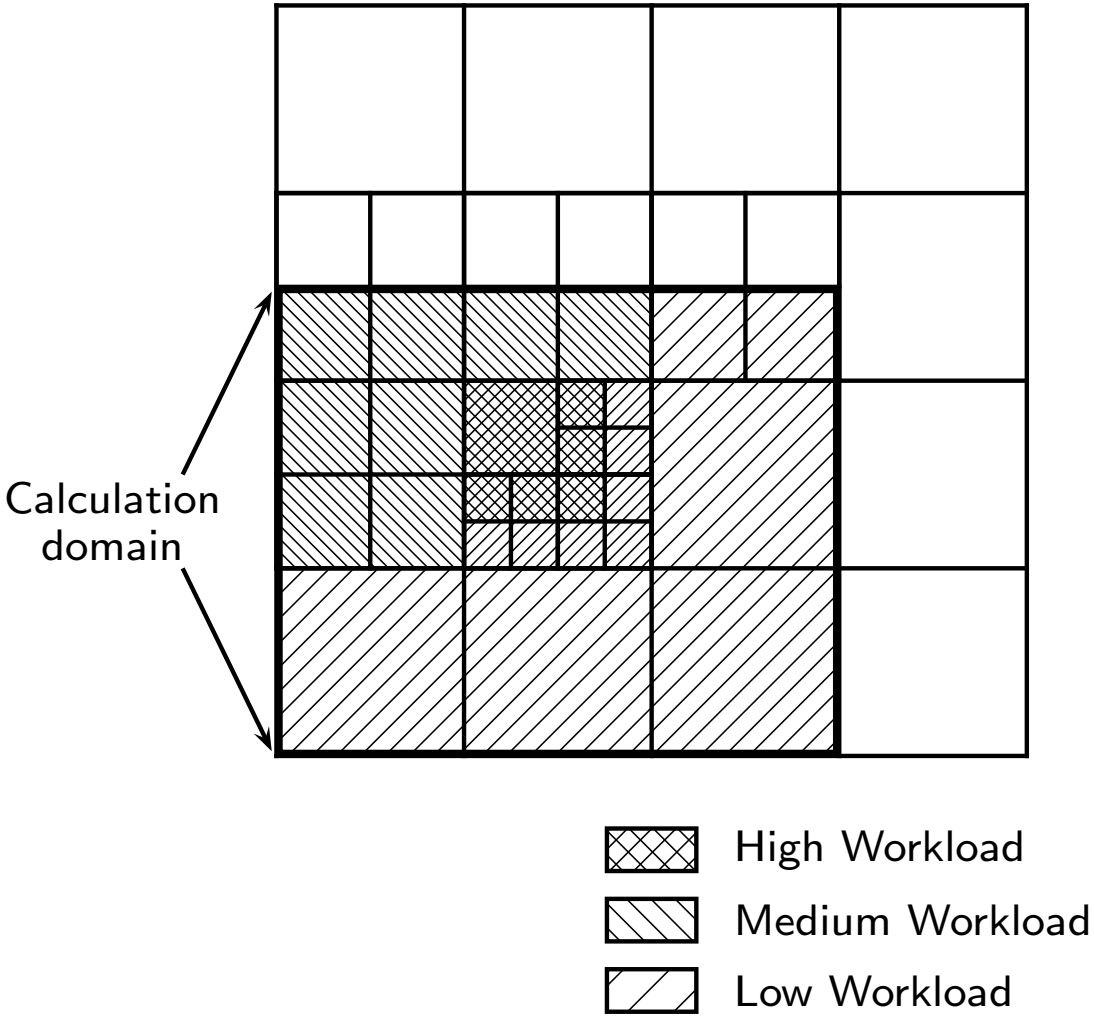
Space-filling curve algorithm



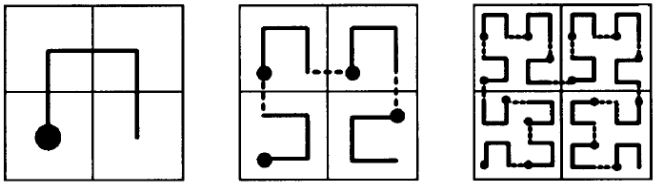


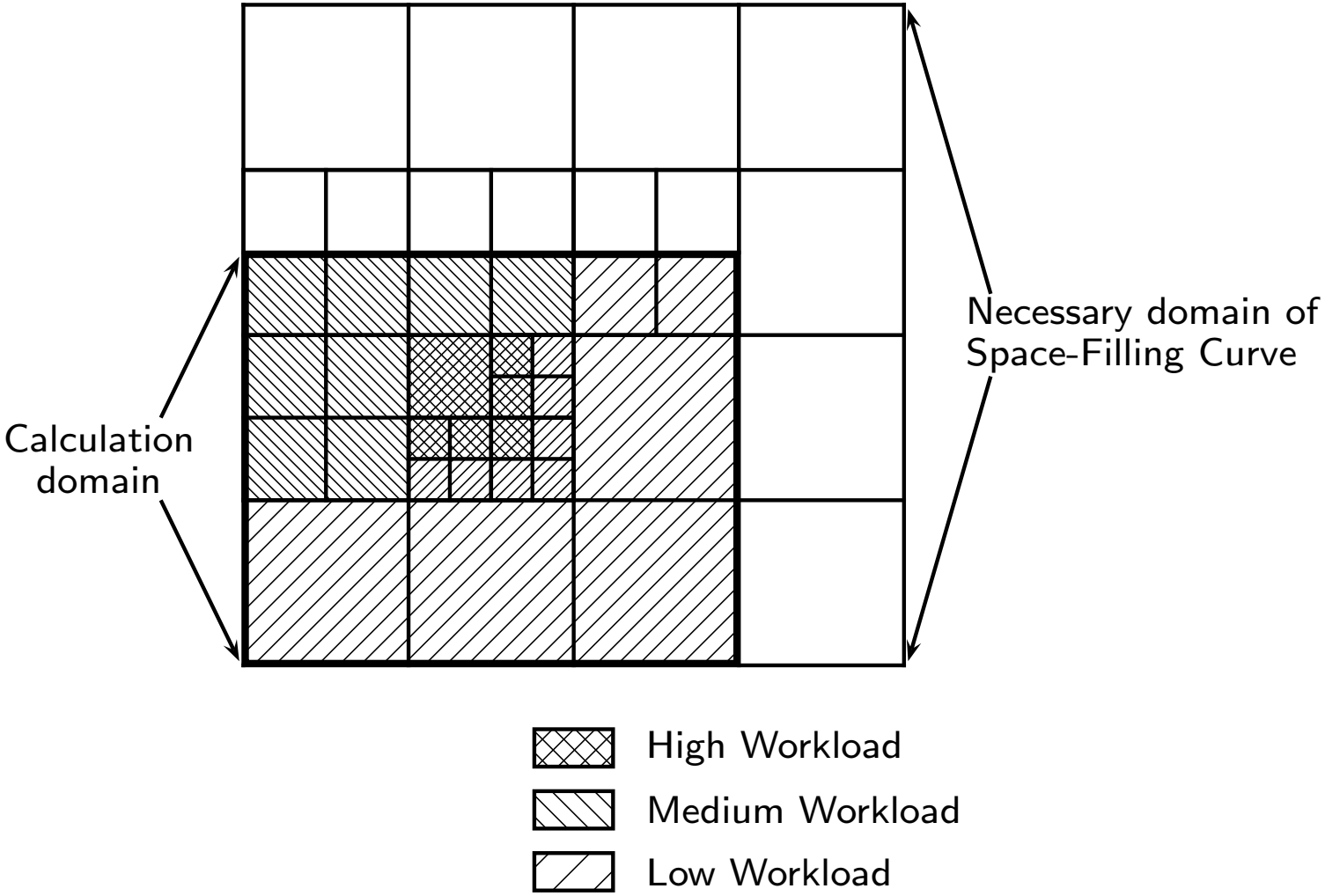
Space-filling curve algorithm



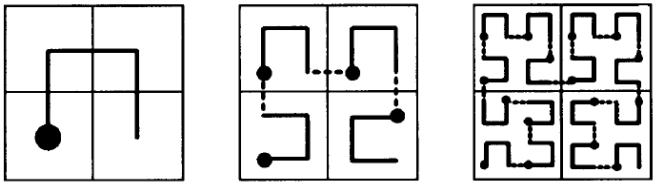


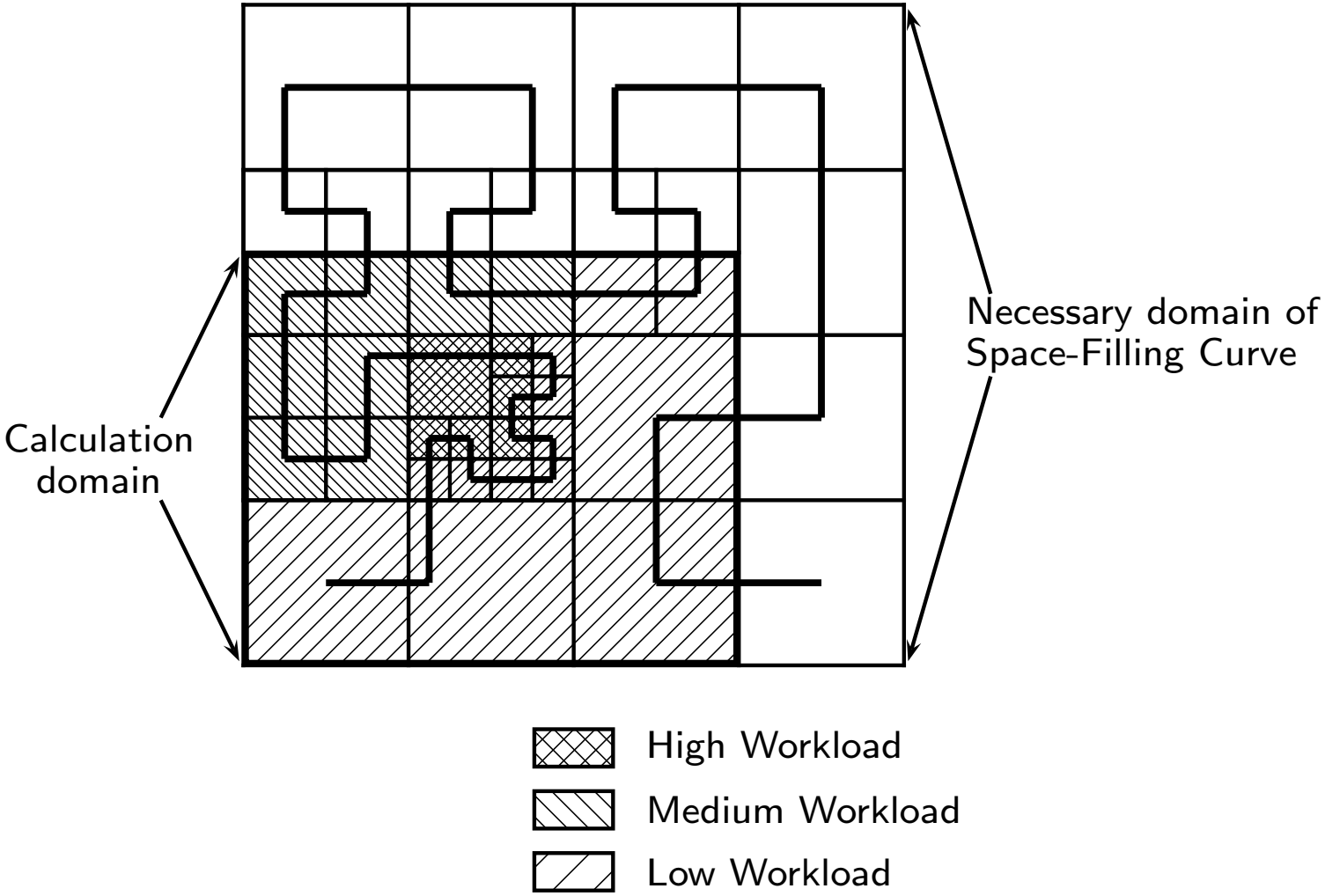
Space-filling curve algorithm



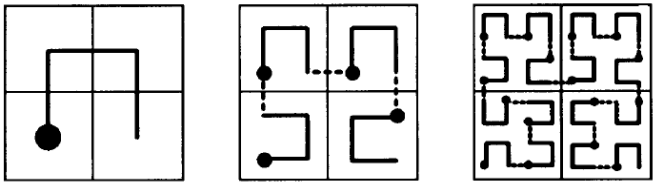


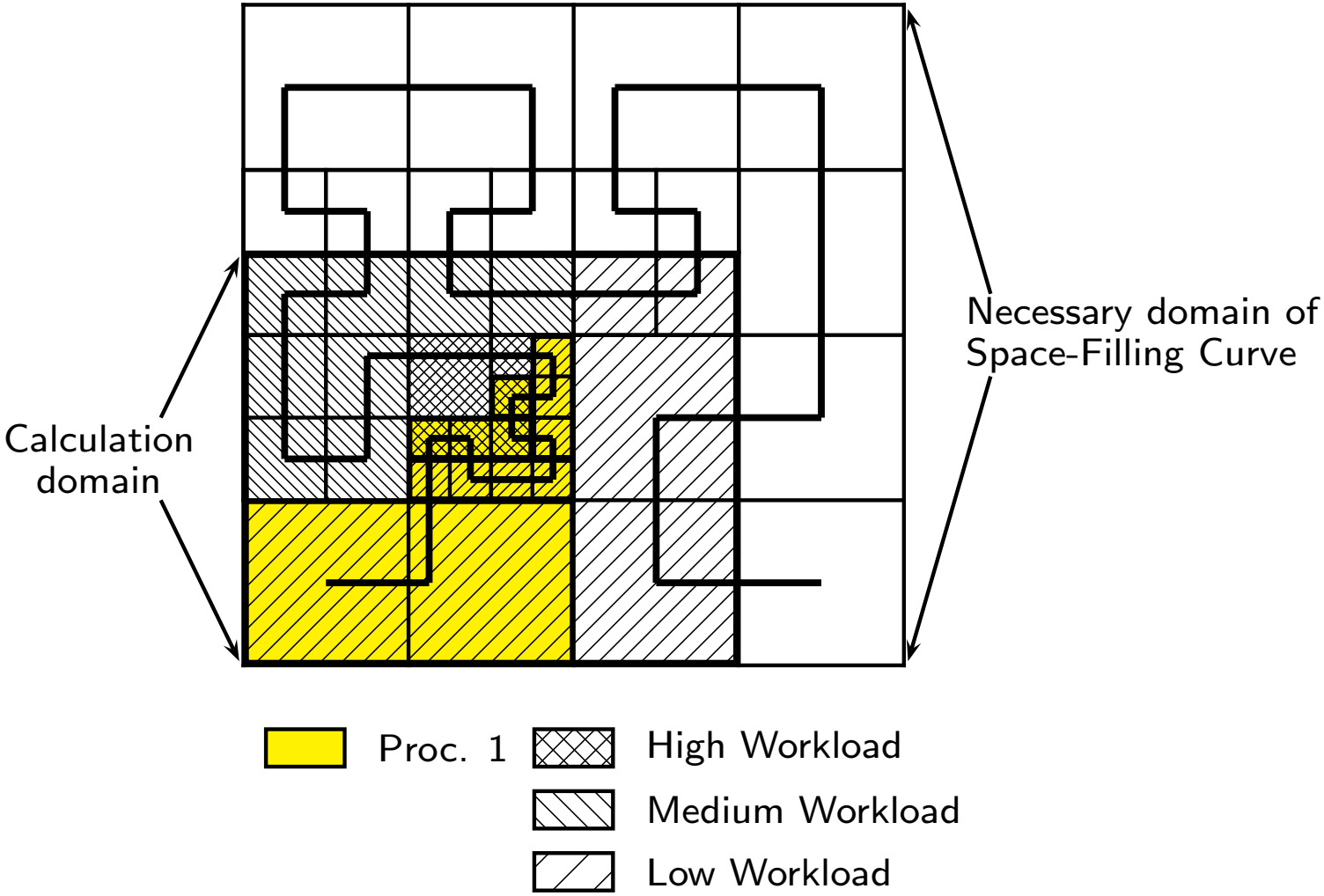
Space-filling curve algorithm



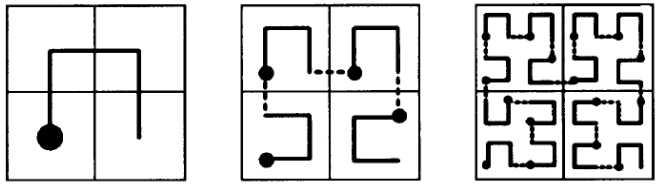


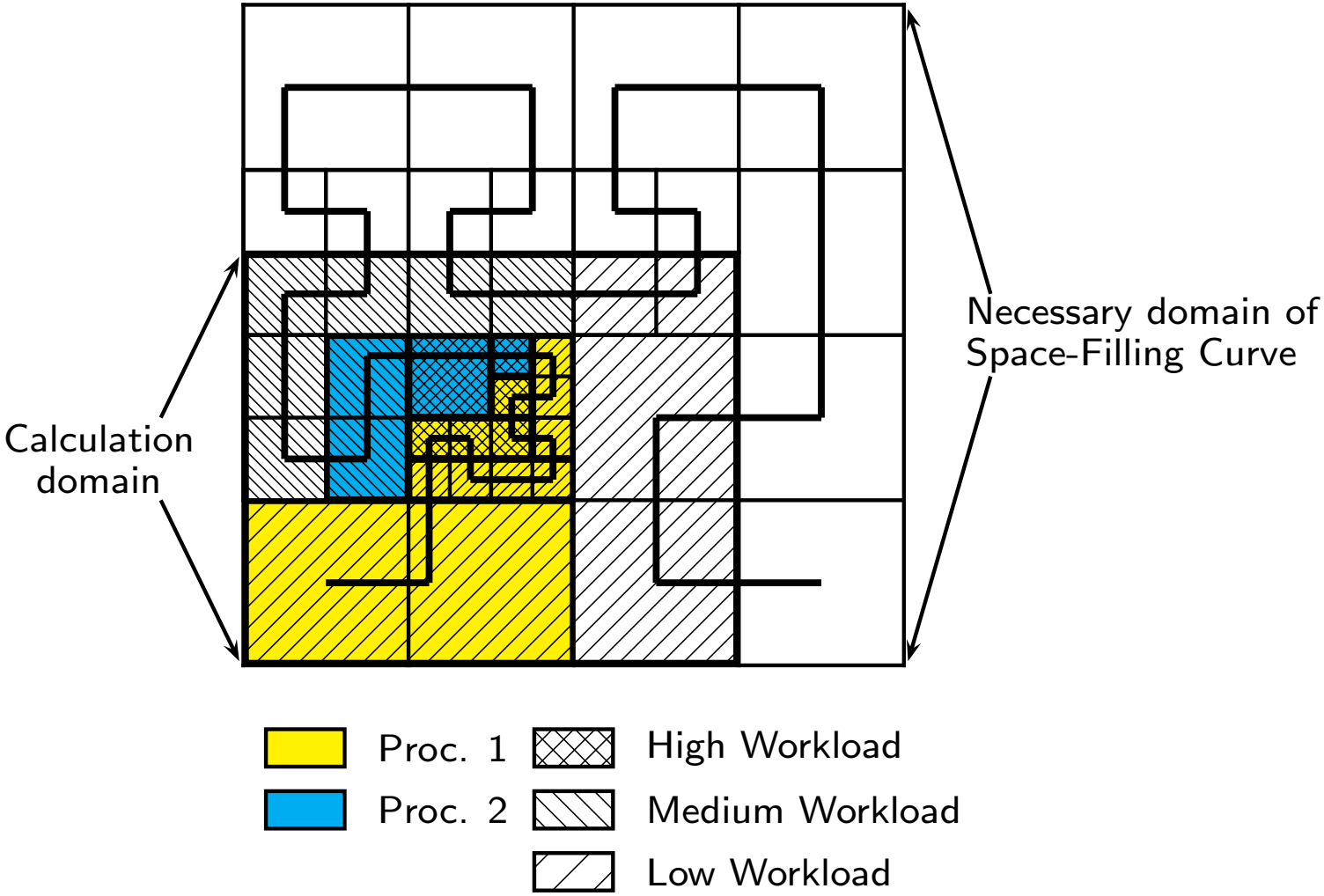
Space-filling curve algorithm



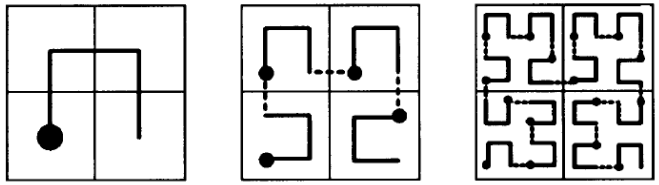


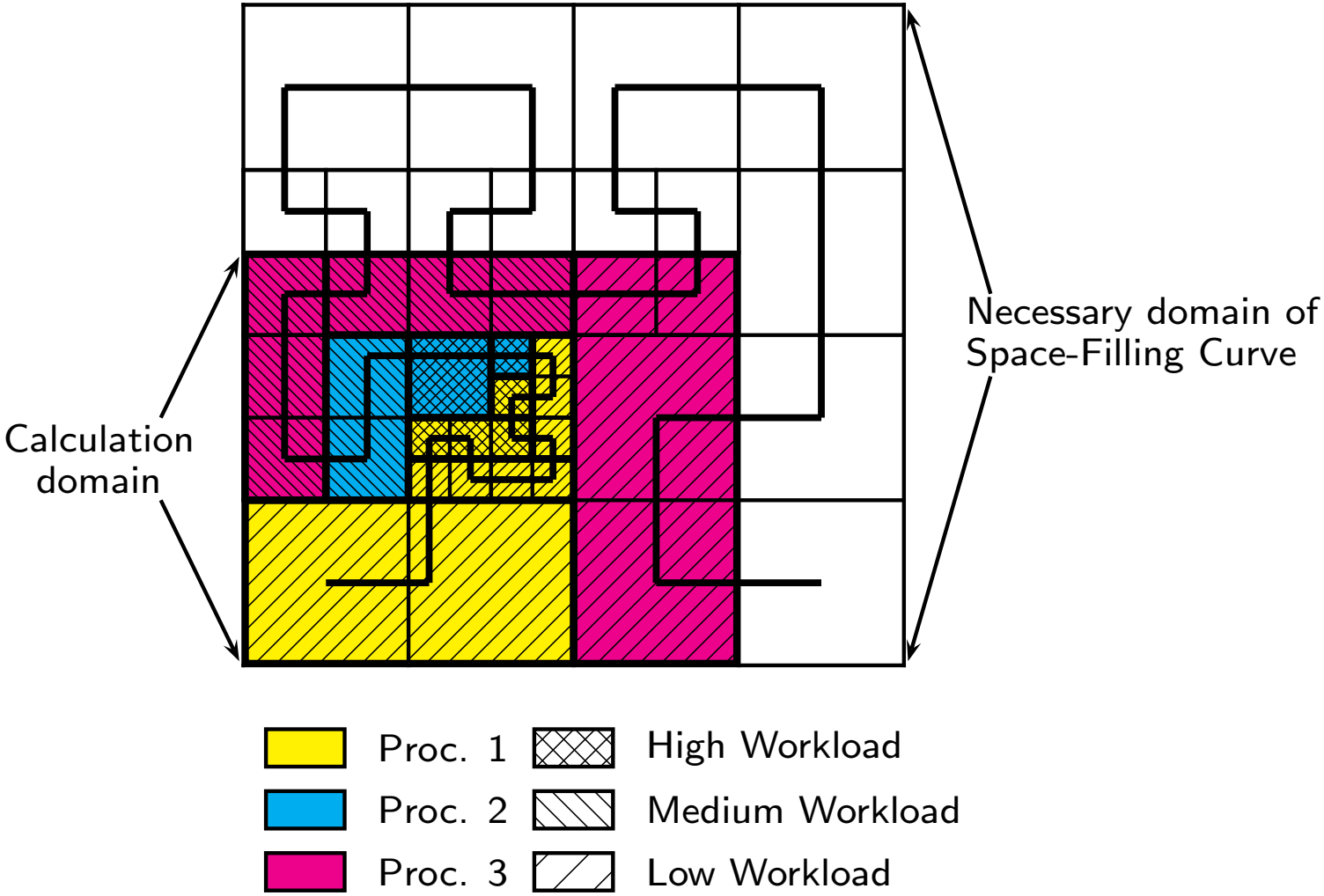
Space-filling curve algorithm





Space-filling curve algorithm





Closest point transform algorithm

The signed distance φ to a surface $\mathcal{I}$ satisfies the eikonal equation [Sethian, 1999]

$$|\nabla\varphi| = 1 \quad \text{with} \quad \varphi\big|_{\mathcal{I}} = 0$$

Solution smooth but non-diferentiable across characteristics.

Closest point transform algorithm

The signed distance φ to a surface $\mathcal{I}$ satisfies the eikonal equation [Sethian, 1999]

$$|\nabla\varphi| = 1 \quad \text{with} \quad \varphi\big|_{\mathcal{I}} = 0$$

Solution smooth but non-diferentiable across characteristics.

Distance computation trivial for non-overlapping elementary shapes but difficult to do efficiently for triangulated surface meshes:

- Geometric solution approach with plosest-point-transform algorithm [Mauch, 2003]

Closest point transform algorithm

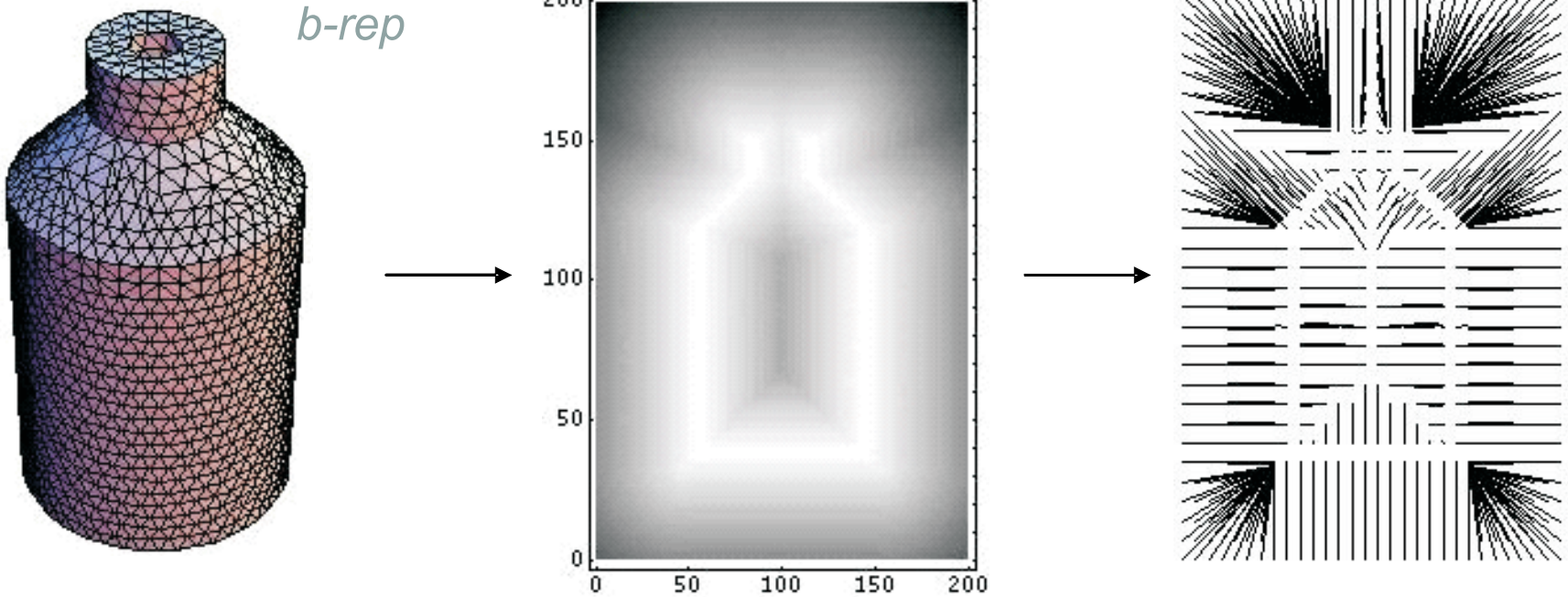
The signed distance φ to a surface $\mathcal{I}$ satisfies the eikonal equation [Sethian, 1999]

$$|\nabla\varphi| = 1 \quad \text{with} \quad \varphi\big|_{\mathcal{I}} = 0$$

Solution smooth but non-diferentiable across characteristics.

Distance computation trivial for non-overlapping elementary shapes but difficult to do efficiently for triangulated surface meshes:

- Geometric solution approach with plosest-point-transform algorithm [Mauch, 2003]

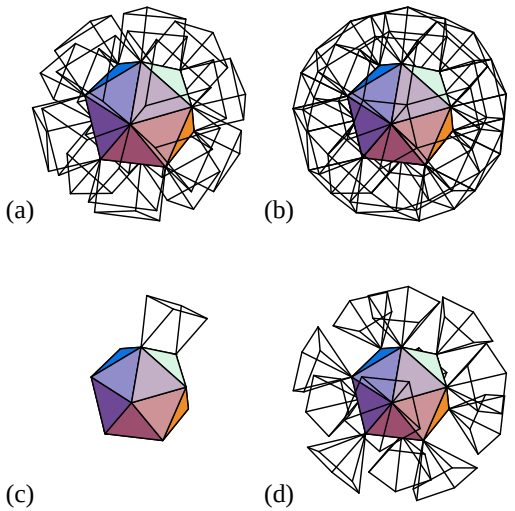


Surface mesh $\mathcal{I}$ Distance φ Normal to closest point

The characteristic / scan conversion algorithm

1. Build the characteristic polyhedrons for the surface mesh

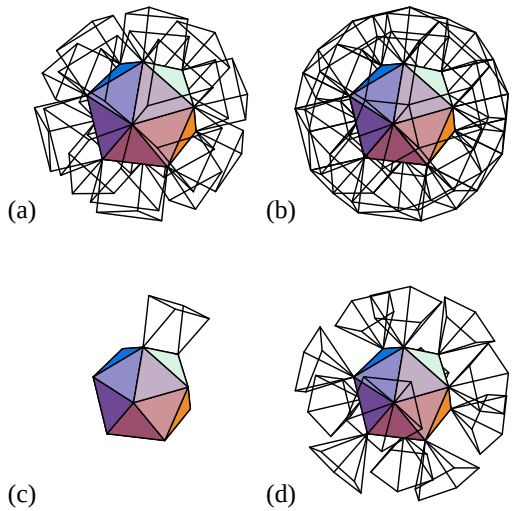
Characteristic polyhedra for faces, edges, and vertices



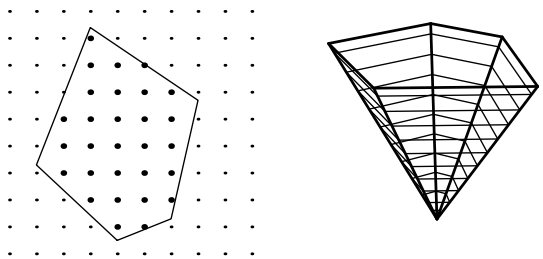
The characteristic / scan conversion algorithm

1. Build the characteristic polyhedrons for the surface mesh
2. For each face/edge/vertex
   2.1 Scan convert the polyhedron.

Characteristic polyhedra for faces, edges, and vertices



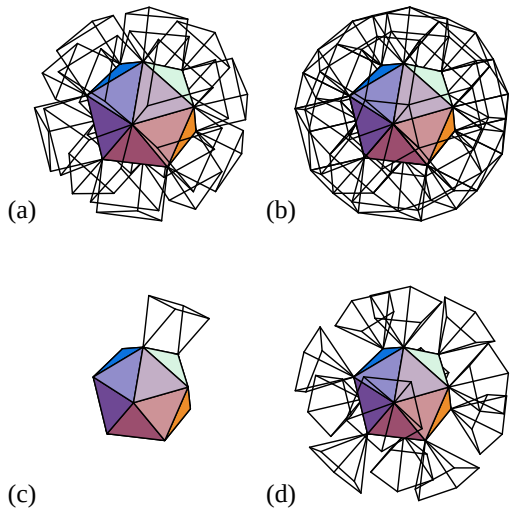
Slicing and scan conversion of apolygon



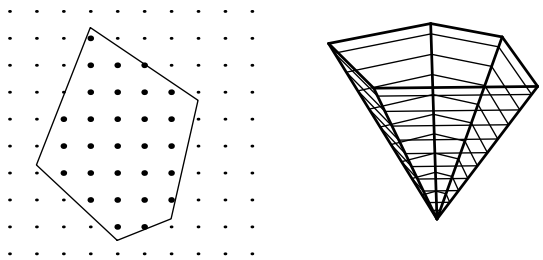
The characteristic / scan conversion algorithm

1. Build the characteristic polyhedrons for the surface mesh
2. For each face/edge/vertex
   - 2.1 Scan convert the polyhedron.
   - 2.2 Compute distance to that primitive for the scan converted points

Characteristic polyhedra for faces, edges, and vertices



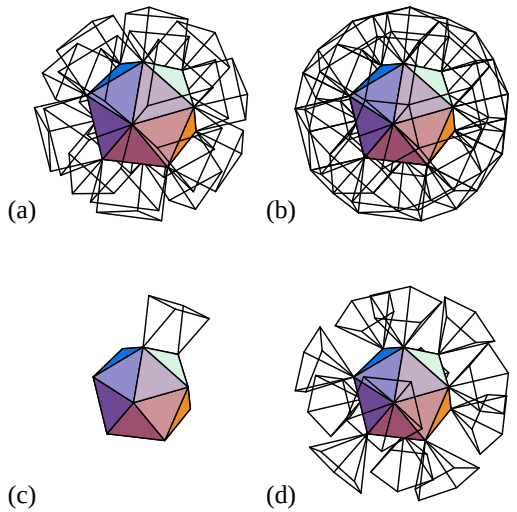
Slicing and scan conversion of apolygon



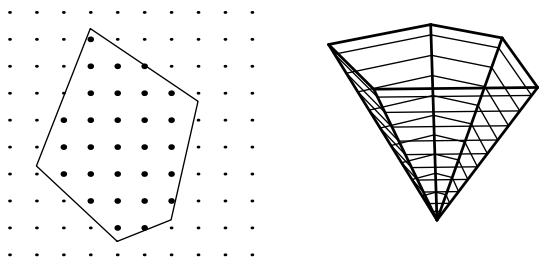
The characteristic / scan conversion algorithm

1. Build the characteristic polyhedrons for the surface mesh
2. For each face/edge/vertex
   - 2.1 Scan convert the polyhedron.
   - 2.2 Compute distance to that primitive for the scan converted points
3. Computational complexity.
   - $O(m)$ to build the b-rep and the polyhedra.
   - $O(n)$ to scan convert the polyhedra and compute the distance, etc.

Characteristic polyhedra for faces, edges, and vertices



Slicing and scan conversion of apolygon

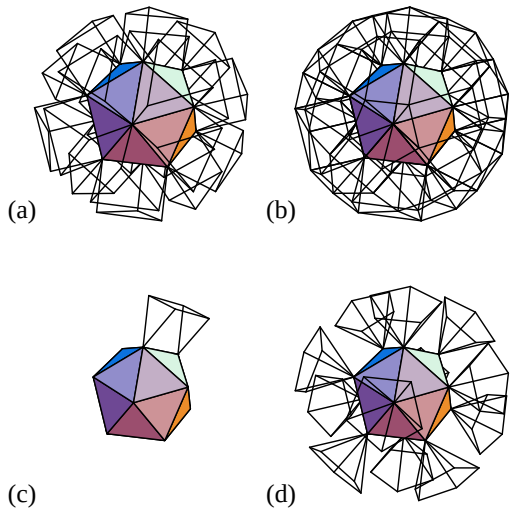


The characteristic / scan conversion algorithm

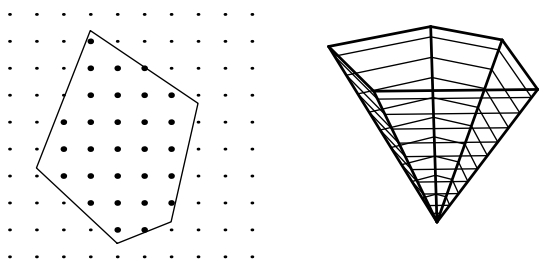
1. Build the characteristic polyhedrons for the surface mesh
2. For each face/edge/vertex
   - 2.1 Scan convert the polyhedron.
   - 2.2 Compute distance to that primitive for the scan converted points
3. Computational complexity.
   - $O(m)$ to build the b-rep and the polyhedra.
   - $O(n)$ to scan convert the polyhedra and compute the distance, etc.
4. Problem reduction by evaluation only within specified max. distance

[Mauch, 2003], see also
[Deiterding et al., 2006]

Characteristic polyhedra for faces, edges, and vertices



Slicing and scan conversion of apolygon



Driven cavity - 3d cavity

- Intel Xeon-2.67 GHz 6-core (Westmere) dual-processor nodes with Qlogics interconnect
- Unigrid with 1 ghost cell

Cores	6	12	24	48	96
Time per step	2.80s	1.46s	0.73s	0.37s	0.18s
Par. Efficiency	100.00%	96.09%	95.33%	95.21%	94.82%
LBM Update	78.05%	77.08%	75.85%	74.50%	71.38%
Synchronization	7.25%	8.67%	10.00%	11.32%	14.35%
Phys. Boundary	0.51%	0.46%	0.45%	0.44%	0.44%
Misc	14.19%	13.79%	13.70%	13.73%	13.83%

- AMR with 4 ghost cells

Cores	6	12	24	48	96
Time per step	3.32s	1.90s	1.21s	0.54s	0.30s
Par. Efficiency	100.00%	87.42%	68.76%	77.02%	68.19%
LBM Update	43.44%	40.93%	31.33%	34.64%	30.11%
Synchronization	14.13%	18.26%	34.73%	25.76%	30.69%
Phys. Boundary	1.03%	0.98%	0.77%	0.86%	0.77%
Regridding	15.53%	16.02%	13.87%	18.72%	20.82%
Interpolation	16.74%	15.71%	11.95%	13.15%	11.51%
Fixup	2.89%	2.60%	2.02%	2.28%	2.03%
Misc	6.22%	5.50%	5.33%	4.59%	4.08%

Driven cavity - 3d cavity

- Intel Xeon-2.67 GHz 6-core (Westmere) dual-processor nodes with Qlogics interconnect
- Unigrid with 1 ghost cell

Cores	6	12	24	48	96
Time per step	2.80s	1.46s	0.73s	0.37s	0.18s
Par. Efficiency	100.00%	96.09%	95.33%	95.21%	94.82%
LBM Update	78.05%	77.08%	75.85%	74.50%	71.38%
Synchronization	7.25%	8.67%	10.00%	11.32%	14.35%
Phys. Boundary	0.51%	0.46%	0.45%	0.44%	0.44%
Misc	14.19%	13.79%	13.70%	13.73%	13.83%

- AMR with 4 ghost cells

Cores	6	12	24	48	96
Time per step	3.32s	1.90s	1.21s	0.54s	0.30s
Par. Efficiency	100.00%	87.42%	68.76%	77.02%	68.19%
LBM Update	43.44%	40.93%	31.33%	34.64%	30.11%
Synchronization	14.13%	18.26%	34.73%	25.76%	30.69%
Phys. Boundary	1.03%	0.98%	0.77%	0.86%	0.77%
Regridding	15.53%	16.02%	13.87%	18.72%	20.82%
Interpolation	16.74%	15.71%	11.95%	13.15%	11.51%
Fixup	2.89%	2.60%	2.02%	2.28%	2.03%
Misc	6.22%	5.50%	5.33%	4.59%	4.08%

- Expense for boundary is increased compared to FV methods because the algorithm uses few floating point operations but a large state vector!