

spaceTimeWindow Library

Technical Manual

Version 1.0 | OpenFOAM v2512

Boundary Data Extraction and Flow Recalculation • Time Interpolation • Integrated Zstd Compression • Optional Encryption

Alin-Adrian Anton

Politehnica University of Timișoara

Last updated: February 9, 2026

OPENFOAM® is a registered trademark of OpenCFD Limited, producer and distributor of the OpenFOAM software via <https://www.openfoam.com>.

This work is licensed under CC BY-SA 4.0.
<https://creativecommons.org/licenses/by-sa/4.0/>

Copyright © 2026 Alin-Adrian Anton

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

<https://creativecommons.org/licenses/by-sa/4.0/>

OPENFOAM® is a registered trademark of OpenCFD Limited.

CHERUBIM® is a registered trademark of the author. This software is libre software (libreware).

If this software has been useful in your research, please consider instructions from the *Acknowledging this work* section.

Contents

1	Introduction	6
1.1	Requirements	6
1.2	Building the Library	6
2	Concept: Space-Time Window Extraction and Reconstruction	7
2.1	The Challenge of Large-Scale CFD Simulations	7
2.2	The Space-Time Window Concept	8
2.3	Extraction and Reconstruction Workflow	8
2.3.1	Extraction Phase (HPC)	8
2.3.2	Reconstruction Phase (Offline)	9
2.4	Parallel Execution with MPI Domain Decomposition	9
2.4.1	How Parallel Extraction Works	11
2.5	Data Storage: Initial Fields and Boundary Data	12
2.5.1	Initial Fields	12
2.5.2	Boundary Data	12
2.6	Why Reconstruction Works	13
2.7	Automatic Boundary Condition Configuration	14
2.7.1	Resolution-Independent Reconstruction	14
2.7.2	Time Interpolation at Runtime	15
3	Encryption of Boundary Data	16
3.1	Key Generation	16
3.2	Secure Key Storage	17
3.3	Encrypted Extraction	17
3.4	Decryption During Case Initialization	18
3.5	Security Properties	18
3.5.1	Cryptographic Foundations	18
3.6	Ethical Considerations and HPC Transparency	19
3.7	Protection Against Malware and Ransomware	19
3.7.1	Secure Archival Strategy	20
3.7.2	Security Model	20
4	Boundary Condition Approaches	20
4.1	Pressure-Coupled BC — Recommended	21
4.2	Precise BC (-preciseBC) — Highest Fidelity Reconstruction	22
4.3	Accuracy Comparison	24
4.4	Inlet-Outlet BC (-inletOutletBC) — For Quick Tests	25
4.5	Fixed Outlet Direction (-outletDirection)	26
4.6	Ghost Cell Mode (-ghostCells) — Machine-Precision Reconstruction	26
4.7	Boundary Condition Assignment Logic	28
5	Workflow Overview	29
5.1	Serial Workflow	29
5.2	Parallel Workflow	30
6	Configuration Reference	30
6.1	Extraction Configuration	30
6.2	Initial Fields vs Boundary Data Fields	32
6.3	Boundary Condition Configuration	33

6.3.1	spaceTimeWindowCoupledPressure (Recommended for Pressure)	33
6.3.2	spaceTimeWindowInletOutlet	35
6.3.3	spaceTimeWindow (Pure Dirichlet)	36
6.4	Case Initialization Options	36
6.5	Mesh Coarsening and Refinement	38
6.5.1	Spatial Interpolation Algorithms	38
6.5.2	Initial Field Interpolation	40
7	Data Storage Format	42
7.1	Directory Structure	42
7.2	Boundary Data Compression	43
7.2.1	Delta-Varint Codec (DVZ)	43
7.2.2	DVZ Mathematical Formulation	43
7.2.3	Delta-Varint-Temporal Codec (DVZT)	45
7.2.4	DVZT Mathematical Formulation	47
7.2.5	Integrated Zstd Compression	48
7.2.6	External Compression Benchmark	49
7.3	Extraction Metadata	50
8	Parallel Execution	51
8.1	Parallel Extraction	51
8.2	Field Reconstruction	51
8.3	Parallel Reconstruction	52
9	Mass Conservation	52
9.1	The Mass Imbalance Problem	52
9.2	Solutions	52
9.2.1	Use Pressure-Coupled BC (Recommended)	52
9.2.2	Use -inletOutletBC (for quick tests)	53
9.2.3	Use -correctMassFlux	53
9.2.4	Use -outletDirection with fixesValue true	53
9.3	The fixesValue Option	53
9.4	Outlet Patch for Pressure Relief	54
9.5	Recommended Configurations	54
10	Solver Settings and Relaxation	55
10.1	Time Stepping	55
10.2	PIMPLE Settings	55
10.3	Relaxation Factors	55
10.4	Linear Solver Settings	56
11	Time Interpolation	56
11.1	Exact Timestep Matching (Default)	56
11.2	Linear Interpolation	58
11.3	Cubic Spline Interpolation	59
11.3.1	Catmull-Rom Spline Formulation	59
11.4	Time Range Requirements	60
12	Flux Reporting and Diagnostics	60
13	Troubleshooting	60
13.1	Common Errors	60

13.1.1	Time Step Mismatch	60
13.1.2	No Matching Timestep	61
13.1.3	Pressure Solver Divergence	61
13.1.4	Encryption Errors	61
14	Acknowledging this work	61
15	Acknowledgments	62
A	Example Case: ERCOFTAC UFR2-02 Square Cylinder	62
A.1	Reference Data and Validation Resources	62
A.2	Physical Background: Von Kármán Vortex Street	63
A.3	Domain and Mesh Configuration	63
A.4	Extraction Box Placement	64
A.5	Physical Parameters	64
A.6	Running the Example	64
A.7	Validation	65
A.7.1	CFD Validation Philosophy	65
A.7.2	Validating the Reconstruction	66
B	Field Selection by Turbulence Model	66
C	The <code>pimpleFoamPrecise</code> Solver	66
C.1	Motivation	67
C.2	Implementation	67
C.3	Usage	68
D	The <code>pimpleFoamStag</code> Solver	68
D.1	Algorithm	68
D.2	Colocated vs Staggered Discretization	69
D.3	Comparison of <code>pimpleFoamPrecise</code> and <code>pimpleFoamStag</code>	70
D.4	Ghost Cell Support	70
E	The <code>ghostCellConstraint fvOption</code>	71
E.1	Configuration	71
E.2	How It Works	71

1 Introduction

The `spaceTimeWindow` library enables extraction and reconstruction of spatial subsets from Large Eddy Simulation (LES) and transient RANS simulations in OpenFOAM. This technique allows researchers to:

- Extract a region of interest from a full-domain simulation
- Store time-varying boundary conditions efficiently with compact codecs (DVZ/DVZT)
- Optionally apply integrated `zstd` compression for additional $\sim 10\times$ size reduction
- Reconstruct the flow within the subset using pre-computed boundary data
- Optionally encrypt boundary data for secure distribution

The library consists of four main components:

1. **`spaceTimeWindowExtract`** — Function object for boundary data extraction during simulation
2. **`spaceTimeWindowInitCase`** — Utility for reconstruction case initialization
3. **`spaceTimeWindow`** — Pure Dirichlet boundary condition for applying extracted data
4. **`spaceTimeWindowInletOutlet`** — Flux-based boundary condition (recommended for unsteady flows)

1.1 Requirements

- OpenFOAM v2512 (openfoam.com) or compatible ESI-OpenCFD version
- C++17 compatible compiler (GCC 7+ or Clang 5+)
- Optional: `libzstd` for integrated `zstd` compression (`libzstd-dev` or `libzstd-devel`)
- Optional: `libsodium` for encryption support (`libsodium-dev` or `libsodium-devel`)
- Optional: MPI for parallel execution

OPENFOAM® is a registered trademark of OpenCFD Limited.

1.2 Building the Library

```
1 # Navigate to source directory
2 cd openfoam-spaceTimeWindow
3
4 # Build library and utilities
5 # Auto-detects libzstd (compression) and libsodium (encryption)
6 ./Allwmake
```

Listing 1: Building `spaceTimeWindow`

Optional dependencies are auto-detected by `./Allwmake`:

- **libzstd**: Enables integrated zstd compression of DVZ/DVZT files (~10× additional reduction)
- **libsodium**: Enables X25519 encryption of boundary data

For manual building with optional features:

```

1 cd src/spaceTimeWindow
2 export FOAM_USE_ZSTD=1 FOAM_USE_SODIUM=1 && wmake
3
4 cd applications/utilities/preProcessing/spaceTimeWindowInitCase
5 export FOAM_USE_ZSTD=1 FOAM_USE_SODIUM=1 && wmake
6
7 cd ../spaceTimeWindowKeygen
8 export FOAM_USE_SODIUM=1 && wmake

```

Listing 2: Manual build with optional features

2 Concept: Space-Time Window Extraction and Reconstruction

This section introduces the fundamental concepts behind space-time window extraction and reconstruction in high performance fluid flow simulations. Understanding these concepts is essential before diving into the technical implementation details.

2.1 The Challenge of Large-Scale CFD Simulations

Modern computational fluid dynamics (CFD) simulations, particularly Large Eddy Simulations (LES) and Direct Numerical Simulations (DNS), generate enormous amounts of data. A typical industrial LES simulation might involve:

- Meshes with 10^7 – 10^9 cells
- Thousands of timesteps ($\delta t = 1^{-6}$ time stepping)
- Multiple field variables (velocity, pressure, turbulence quantities)
- Terabytes to petabytes of raw output data

Quite often the simulation domain is large enough to allow for the fluid flow to fully develop within much smaller local regions of interest. The European Research Community on Flow, Turbulence and Combustion (ERCOTAC) database [1] contains numerical results and laboratory measurements, and new software implementations must be validated against previously confirmed results (physics remains the same, numerical solvers may change).

Nevertheless, validation is always performed in relation to the physics of the fluid flow, not against bit-exact numerical values from previous simulations.

Storing and post-processing this data is often impractical. When a researcher defines a *region of interest* within a large fluid flow domain, the goal is to **zoom in** and obtain richer detail about the flow in that region—higher spatial or temporal resolution, more detailed post-processing, or the ability to explore “what-if” scenarios with different turbulence models—not to produce a bit-by-bit identical copy of the internal fields from the original simulation. The space-time

window approach addresses precisely this need: it extracts only the boundary data required to drive an independent simulation within the region of interest at a later time, and allows the user to change the computational resolution, numerical schemes, or post-processing strategy “at home”, on modest hardware. The reconstructed flow is physically faithful—it satisfies the same governing equations with the same boundary conditions—but it is not, and need not be, numerically identical to the original (see section 2.6 for a discussion of the discretisation differences).

2.2 The Space-Time Window Concept

A **space-time window** [2–4] is a bounded region in both space (a 3D box) and time (a time interval) from which we extract the information necessary to reconstruct the flow field. The key insight is that we only need:

1. **Initial conditions:** The complete field state at one timestep (t_0)
2. **Boundary data:** Time-varying values on the boundary of the spatial region

Figure 1 illustrates this concept. The extraction box (blue) defines a spatial subset of the full domain (gray). During simulation, we record the field values on the boundary of this box at every timestep, creating a complete “boundary history” that can later drive a reconstruction simulation.

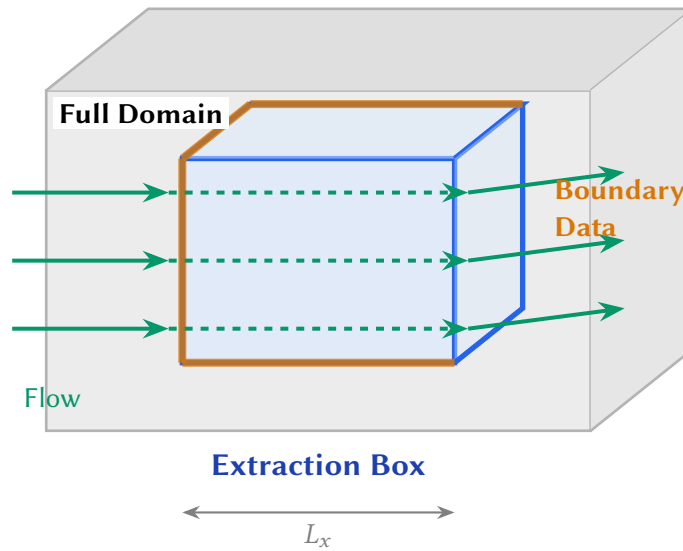


Figure 1: Space-time window extraction: A bounding box (blue) defines the region of interest within the full computational domain (gray). Boundary data (orange) is recorded at every timestep as the flow passes through.

2.3 Extraction and Reconstruction Workflow

The workflow consists of two distinct phases that can be separated in time and computational resources, as shown in fig. 2.

2.3.1 Extraction Phase (HPC)

During the original simulation on high-performance computing resources:

1. The `spaceTimeWindowExtract` function object monitors the simulation

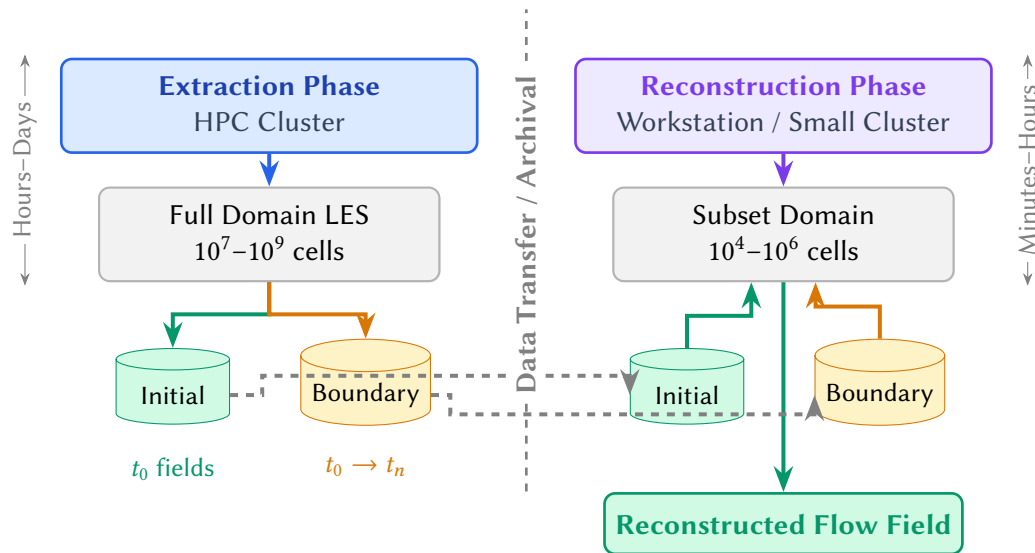


Figure 2: Two-phase workflow: The extraction phase runs on HPC resources and produces compact data files. The reconstruction phase can run on modest hardware, reproducing the flow in the region of interest.

2. At the start time (t_0), it extracts the complete field state within the bounding box
3. At every timestep, it interpolates field values to the boundary faces and stores them
4. The output is a compact dataset: initial conditions + time-varying boundary data

2.3.2 Reconstruction Phase (Offline)

Later, on more modest computational resources:

1. The `spaceTimeWindowInitCase` utility creates a complete simulation case
2. The subset mesh is extracted from the original
3. Boundary conditions are configured to read the stored boundary data
4. The solver runs, reproducing the flow within the extraction box

✓ Tip

The reconstruction phase typically requires 100–10,000× fewer computational resources than the original simulation, making it feasible to run on a workstation or laptop.

2.4 Parallel Execution with MPI Domain Decomposition

Large-scale CFD simulations invariably run in parallel using MPI (Message Passing Interface). The computational domain is divided among multiple processors, each handling a subset of the mesh. The `spaceTimeWindow` library handles both serial and parallel execution transparently.

Figure 3 illustrates how domain decomposition affects the extraction process. The extraction box may span multiple processor domains, requiring coordination to gather boundary data from all relevant processors.

Parallel Extraction: Domain Decomposition with 8 MPI Processes

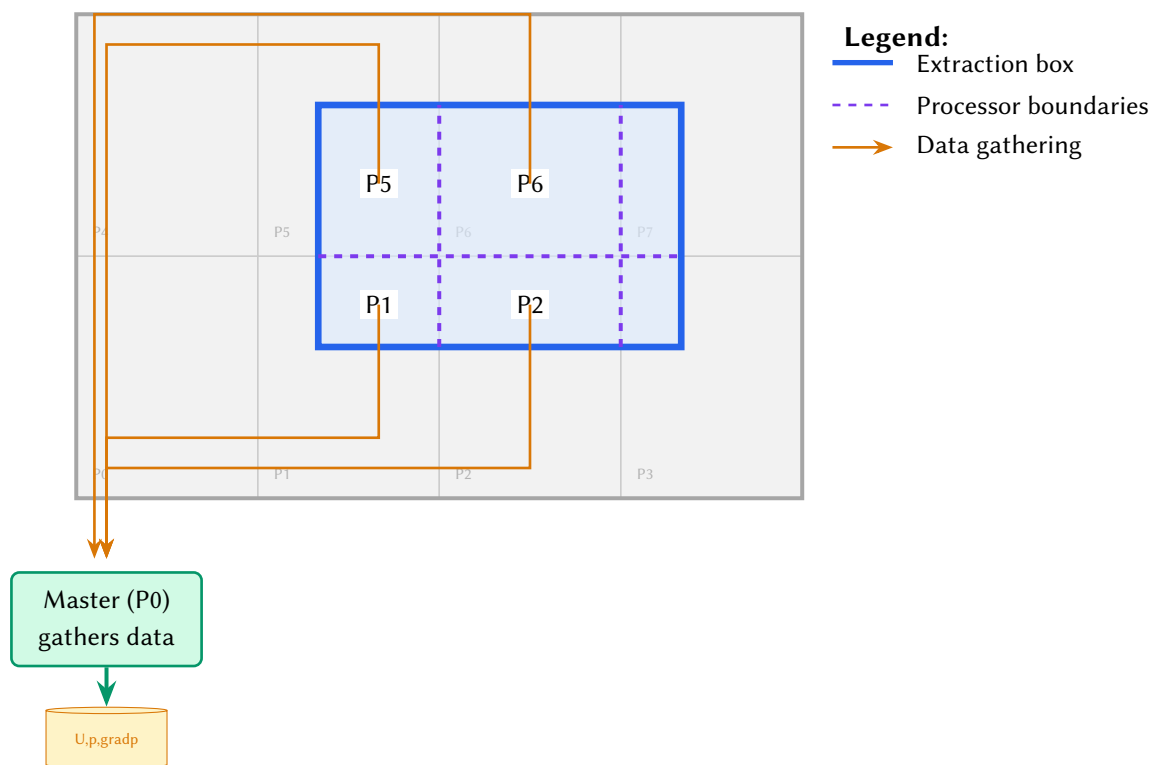


Figure 3: Domain decomposition in parallel simulations. The extraction box (blue) spans multiple processor domains. Each processor extracts its local contribution, and data is gathered to the master process for writing unified boundary data files.

2.4.1 How Parallel Extraction Works

In parallel mode, the extraction process operates as follows:

1. **Local identification:** Each processor identifies which of its cells fall within the extraction bounding box
2. **Local extraction:** Each processor extracts field values for its local portion of the boundary
3. **Global gathering:** Boundary data is gathered to the master processor using MPI collective operations
4. **Unified output:** The master processor writes a single, consolidated boundary data file for each timestep

Figure 4 shows the detailed data flow during the extraction process. The `spaceTimeWindowExtract` function object runs on each MPI process, extracting local contributions that are then gathered and merged into unified output files.

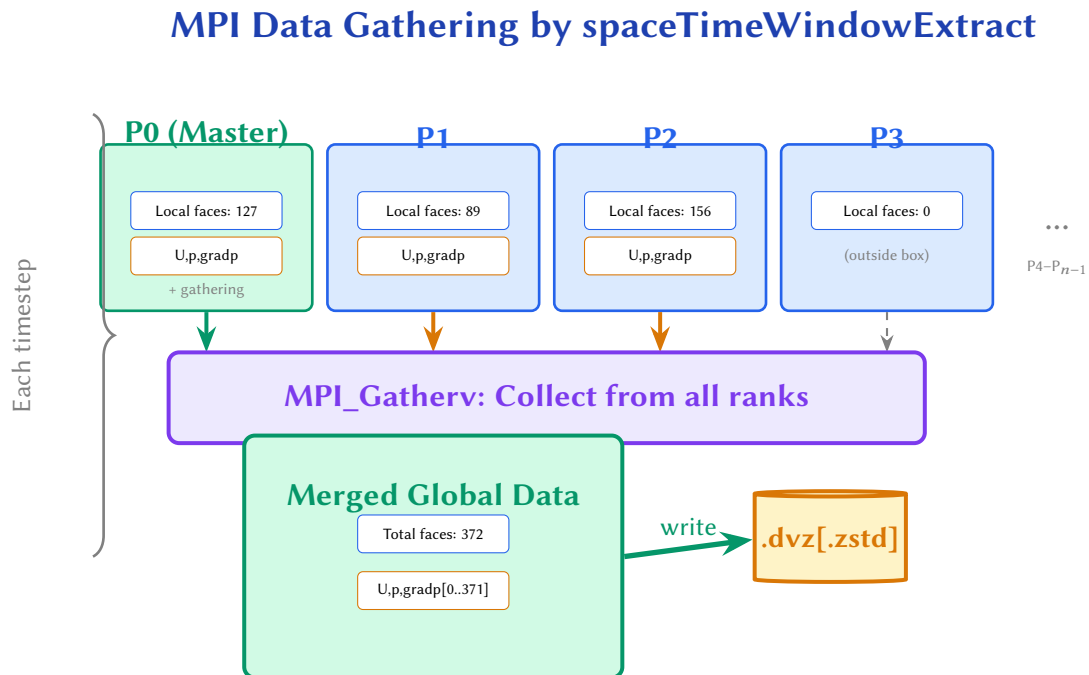


Figure 4: Detailed MPI data gathering during parallel extraction. Each processor runs the `spaceTimeWindowExtract` function object, which identifies local boundary faces, interpolates field values to face centers, and participates in a collective gather operation. The master processor (P0) writes the merged data to a single file per timestep.

This approach ensures that:

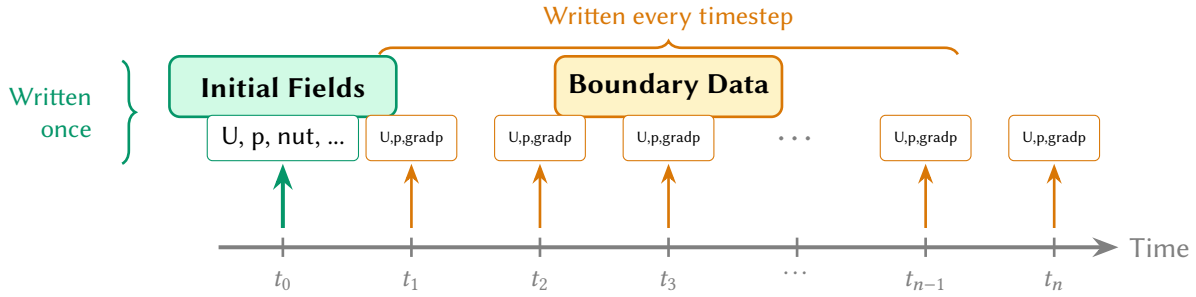
- The reconstruction case is **decomposition-independent**—the same boundary data works regardless of how the original simulation was parallelized
- Output files are compact and self-contained, with no processor-specific information
- The reconstruction can run in serial or with a different parallel decomposition

i Note

In parallel mode, mesh information is not written during extraction. Instead, the extraction box parameters are stored, and `spaceTimeWindowInitCase` creates the subset mesh from reconstructed fields. Run `reconstructPar -time <startTime>` on the source case before initializing the reconstruction case.

2.5 Data Storage: Initial Fields and Boundary Data

The extracted data consists of two distinct components, illustrated in fig. 5.



Storage Requirements:

	Initial fields: $O(N_{\text{cells}})$
	Boundary data: $O(N_{\text{faces}} \times N_{\text{timesteps}})$

→ Boundary data dominates storage. Use dvzt format with integrated zstd for 300×+ compression.

Figure 5: Data storage structure: Initial fields are written once at t_0 , while boundary data is accumulated at every timestep. Compression (DVZ/DVZT + integrated zstd) is critical for the boundary data component.

2.5.1 Initial Fields

The initial fields provide the starting state for reconstruction:

- **What:** Complete volumetric data (U, p, turbulence quantities) within the extraction box
- **When:** Written once at the extraction start time (t_0)
- **Format:** Standard OpenFOAM field format (ASCII)
- **Size:** Proportional to number of cells in the subset, typically megabytes

2.5.2 Boundary Data

The boundary data provides the time-varying boundary conditions:

- **What:** Face-interpolated field values on the boundary of the extraction box
- **When:** Written at every timestep throughout the simulation
- **Format:** Configurable (ASCII, binary, or compressed dvz/dvzt)
- **Size:** Proportional to (number of boundary faces) × (number of timesteps)

△ Warning

Boundary data accumulates rapidly during long simulations. For a typical LES with 10^4 boundary faces and 10^4 timesteps, uncompressed ASCII data can reach tens of gigabytes. The `dvzt` compression format reduces this by 30–50×, and with integrated `zstd` compression the reduction reaches 300×+.

2.6 Why Reconstruction Works

The reconstruction is possible because the governing equations (Navier-Stokes for incompressible flow) are well-posed with appropriate boundary conditions. Mathematically, if we denote:

- $\Omega \subset \mathbb{R}^3$: the extraction box domain
- $\partial\Omega$: the boundary of the extraction box
- $\mathbf{u}(\mathbf{x}, t)$: the velocity field
- $p(\mathbf{x}, t)$: the pressure field

Then knowing $\mathbf{u}(\mathbf{x}, t_0)$ for $\mathbf{x} \in \Omega$ (initial condition) and $\mathbf{u}(\mathbf{x}, t)$ for $\mathbf{x} \in \partial\Omega$, $t \in [t_0, t_n]$ (boundary data) is sufficient to uniquely determine $\mathbf{u}(\mathbf{x}, t)$ for all $\mathbf{x} \in \Omega$, $t \in [t_0, t_n]$.

The reconstruction closely reproduces the original solution when:

1. Time interpolation is disabled (`timeInterpolationScheme none`)
2. The reconstruction runs with the same timestep as extraction
3. The same solver settings and numerical schemes are used
4. The initial fields are stored in binary (lossless) format

i Note

Even with exact boundary data and identical solver settings, the subdomain solution is *not* bit-identical to the full-domain solution. This is a fundamental consequence of the finite volume discretisation: at internal faces, fluxes are computed from two neighbouring cell values using the full computational stencil, whereas at boundary faces with prescribed Dirichlet values, the flux uses the boundary value directly and the matrix contribution changes from an off-diagonal coefficient to a source term [5, 6]. In particular:

- **TVD/high-resolution convective schemes** compute a limiter ratio from upstream and downstream cell values; at boundary faces the upstream stencil is truncated, altering the limiter behaviour [5, 7].
- **Multigrid pressure solvers** (e.g. GAMG) agglomerate cells into coarser levels; the coarsening pattern and the coarse-level matrix structure differ near Dirichlet boundaries compared to the interior [7].
- **Gradient reconstruction** (e.g. Gauss or least-squares) at boundary-adjacent cells uses the prescribed face value instead of a neighbour cell value, changing the gradient estimate [6].

These effects are localised: errors are largest in cells adjacent to the extraction boundary and decay rapidly with distance (see section 4.3). The `-ghostCells` option (section 4.6) eliminates this stencil mismatch entirely by keeping the original extraction boundary faces as internal faces, pushing the mismatch to an outer ghost layer where values are overwritten from extracted data.

With time interpolation enabled (recommended for practical use), the reconstruction closely approximates the original solution, with additional errors controlled by the interpolation scheme (linear or cubic Catmull-Rom spline).

2.7 Automatic Boundary Condition Configuration

A key feature of the `spaceTimeWindow` library is the **automatic configuration** of boundary conditions during offline reconstruction. When you run `spaceTimeWindowInitCase`, the utility analyzes the extracted data and configures appropriate boundary conditions for each field—no manual setup required.

Figure 6 illustrates the automatic configuration process that occurs when setting up a reconstruction case on a laptop or desktop computer.

Automatic BC Configuration by `spaceTimeWindowInitCase`

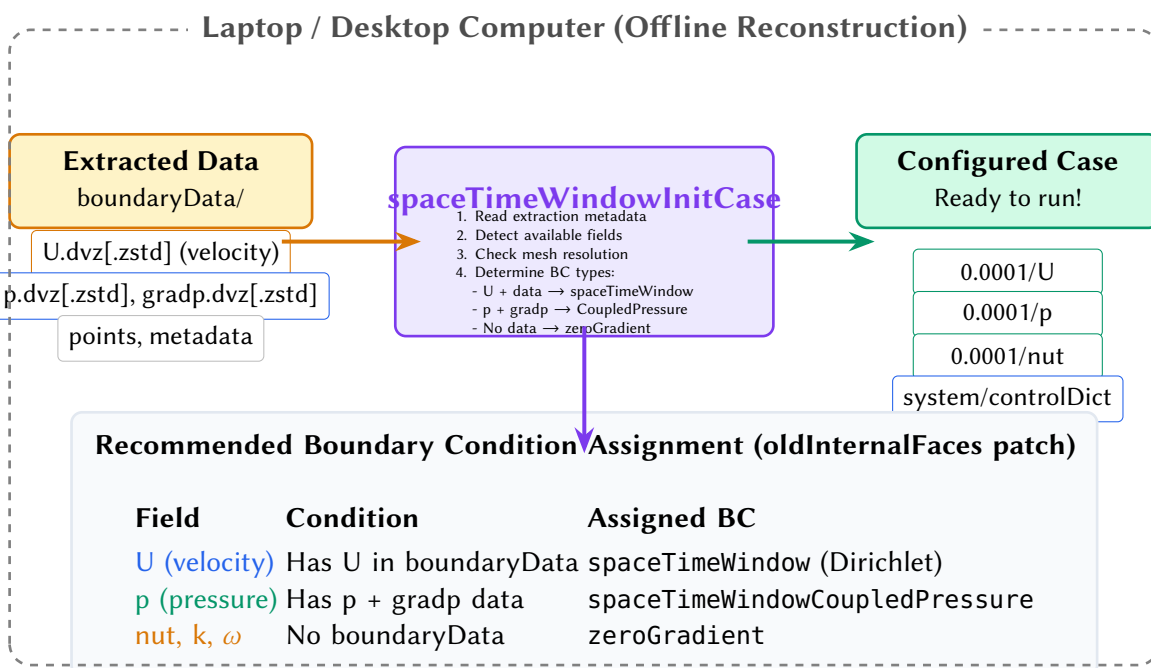


Figure 6: Recommended boundary condition configuration for pressure-coupled reconstruction. The `spaceTimeWindowInitCase` utility can assign BCs based on available data: `spaceTimeWindow` for velocity (Dirichlet), `spaceTimeWindowCoupledPressure` for pressure (mixed Dirichlet/Neumann), and `zeroGradient` for turbulence fields.

2.7.1 Resolution-Independent Reconstruction

The reconstruction can run at **different mesh resolutions** than the original extraction. This enables several important use cases:

- **Coarser mesh:** Faster turnaround for exploratory analysis or debugging
- **Finer mesh:** Higher resolution studies without re-running the expensive global simulation
- **Same resolution:** Highest-fidelity reconstruction for validation

Figure 7 shows how the `spaceTimeWindow` boundary condition automatically handles resolution differences at runtime.

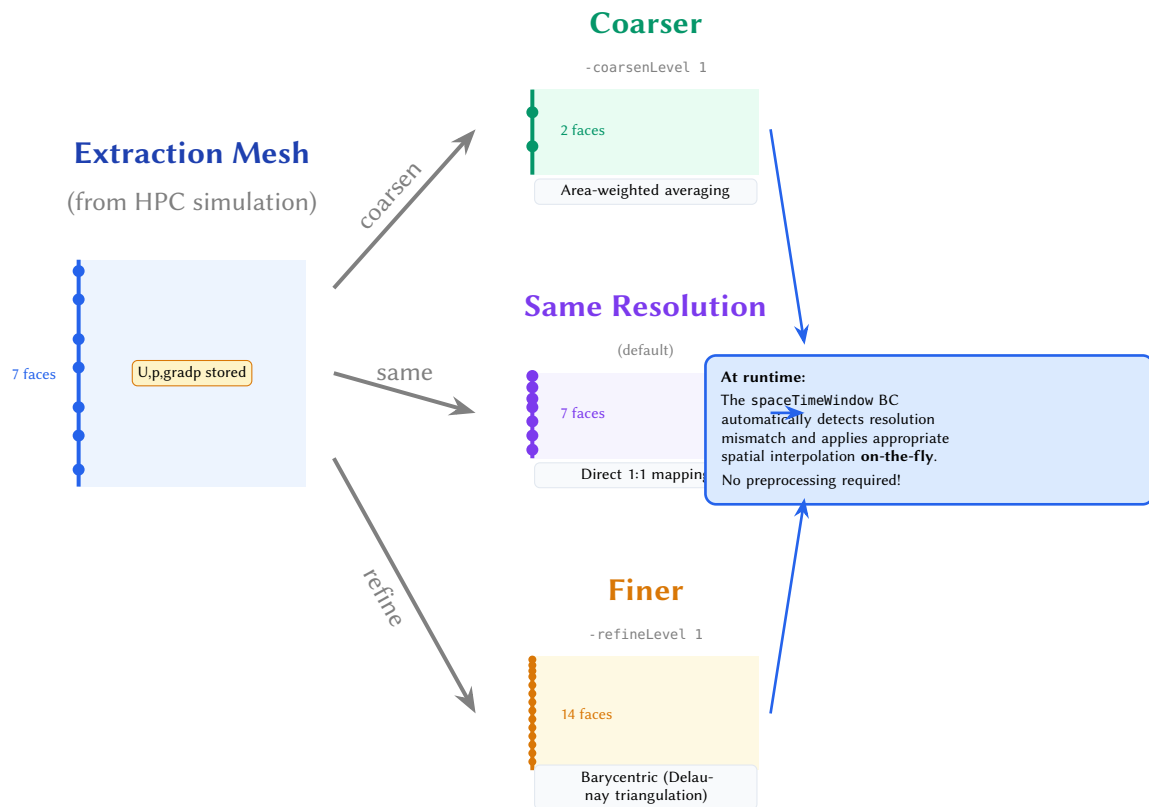


Figure 7: Resolution-independent reconstruction: The `spaceTimeWindow` boundary condition automatically handles mesh resolution differences between extraction and reconstruction. Spatial interpolation is performed on-the-fly during the simulation.

2.7.2 Time Interpolation at Runtime

Similarly, the boundary condition handles **time interpolation** automatically when the reconstruction timestep differs from the extraction timestep (which is common with adaptive timestepping). Figure 8 illustrates the temporal interpolation process.

✓ Tip

The combination of automatic spatial and temporal interpolation means you can:

- Run reconstruction on a laptop with a coarser mesh for quick visualization
- Use adaptive timestepping (`adjustTimeStep yes`) without worrying about timestep alignment

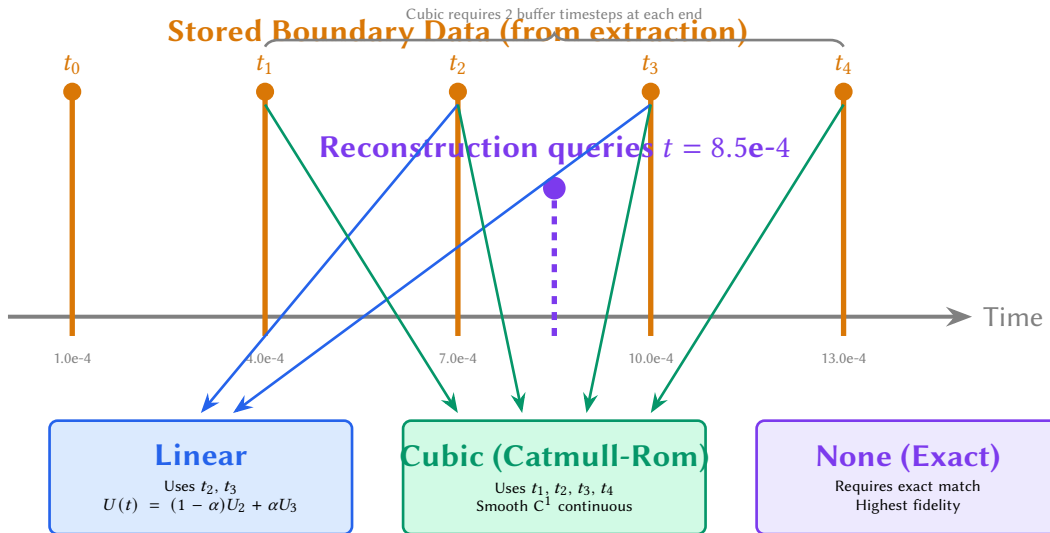


Figure 8: Time interpolation during reconstruction. When the solver requests field values at times not exactly matching stored timesteps, the boundary condition interpolates using linear (2 points) or cubic Catmull-Rom (4 points) schemes. Cubic interpolation provides smooth derivatives, essential for adaptive timestepping.

- Later refine the mesh for detailed analysis without re-extracting data

All interpolation happens transparently at runtime—no preprocessing steps required.

3 Encryption of Boundary Data

The spaceTimeWindow library supports X25519 asymmetric encryption using libsodium sealed boxes. This allows secure distribution of simulation data where:

- Extraction uses only the **public key**
- Reconstruction requires the **private key**
- Data cannot be decrypted without the private key

3.1 Key Generation

Generate a keypair using the provided utility:

```
1 spaceTimeWindowKeygen
2 # Output:
3 # Public key: fqzYQ0U8j27tFEr5WzEMylbvXYP+9CAyk0JhwwZ2rwg=
4 # Private key: *****H8exb*****n4Kv0nu5gqljI2RPBCwl4=
5 #
6 # IMPORTANT: Store the private key securely!
```

Listing 3: Generating encryption keys

△ Warning

The private key must be stored securely and never committed to version control. Anyone with the private key can decrypt all boundary data encrypted with the corresponding public key.

3.2 Secure Key Storage

The private key should be stored using one of the following methods:

- **Hardware security keys:** YubiKey, Nitrokey, or similar FIDO2/PIV-capable devices provide the strongest protection. The private key never leaves the hardware token.
- **Secure password managers:** KeePassXC, 1Password, Bitwarden, or similar tools with strong master passwords and optional hardware key integration.
- **Encrypted vaults:** GPG-encrypted files or OS keychains (macOS Keychain, GNOME Keyring, Windows Credential Manager).

✓ Tip

For maximum security, generate the keypair on an air-gapped machine, store the private key on a hardware token, and only transfer the public key to the HPC system.

3.3 Encrypted Extraction

Add the public key to your extraction configuration in `system/controlDict`:

```

1  functions
2  {
3      extractSubset
4      {
5          type          spaceTimeWindowExtract;
6          libs          (spaceTimeWindow);
7
8          box           ((0.05 -0.25 0.01) (0.90 0.25 0.38));
9          outputDir     "../subset-case";
10         fields        (U p nut);
11
12         writeFormat    deltaVarint;
13
14         // Enable encryption with public key
15         publicKey      "fqzYQ0U8j27tFEr5WzEMylbvXYP+9CAyk0JhwwZ2rwg=";
16
17         writeControl    timeStep;
18         writeInterval   1;
19     }
20 }
```

Listing 4: Encrypted extraction configuration

Encrypted files have the `.enc` extension appended to the codec extension (e.g., `U.dvz.enc`, or `U.dvz.zstd.enc` when integrated `zstd` compression is also enabled).

3.4 Decryption During Case Initialization

When running `spaceTimeWindowInitCase` on encrypted data, you will be prompted for the private key:

```
1 cd subset-case
2 spaceTimeWindowInitCase -sourceCase ../source-case
3 # Enter private key (base64): [input not echoed]
```

Listing 5: Decrypting boundary data

The utility automatically:

1. Derives the public key from the private key
2. Decrypts all `.enc` files in `boundaryData`
3. Removes encrypted files after successful decryption

3.5 Security Properties

- **Sealed box encryption:** Anonymous sender, only recipient can decrypt
- **X25519 key exchange:** 128-bit security level
- **XSalsa20-Poly1305:** Authenticated encryption
- **No plaintext on disk:** Encryption occurs before writing

3.5.1 Cryptographic Foundations

The encryption uses elliptic curve Diffie-Hellman on Curve25519. Key generation produces a private scalar s and public point (eq. (1)):

$$P_{\text{pub}} = s \cdot G \quad (1)$$

where G is the curve's base point. The discrete logarithm problem makes recovering s from P_{pub} computationally infeasible.

Sealed Box Construction For each encryption, a sealed box generates an ephemeral keypair $(e, e \cdot G)$ and computes a shared secret:

$$K = \text{BLAKE2b}(e \cdot P_{\text{recv}}) = \text{BLAKE2b}(e \cdot s_{\text{recv}} \cdot G) \quad (2)$$

The ciphertext is constructed as shown in eq. (3):

$$C = (e \cdot G) \parallel \text{XSalsa20-Poly1305}_K(M) \quad (3)$$

where M is the plaintext boundary data and \parallel denotes concatenation. The ephemeral public key $(e \cdot G)$ is prepended to allow decryption.

Decryption The recipient computes the same shared secret using their private key (eq. (4)):

$$K = \text{BLAKE2b}(s_{\text{recv}} \cdot (e \cdot G)) \quad (4)$$

By the associativity of scalar multiplication on elliptic curves (eq. (5)):

$$s_{\text{recv}} \cdot (e \cdot G) = e \cdot (s_{\text{recv}} \cdot G) = e \cdot P_{\text{recv}} \quad (5)$$

This ensures both parties derive the same key K without ever transmitting it.

Security Level X25519 provides approximately 128 bits of security. Breaking the encryption requires either:

- Solving the elliptic curve discrete logarithm problem (ECDLP): $O(2^{128})$ operations
- Brute-forcing the symmetric key: $O(2^{256})$ operations for XSalsa20

3.6 Ethical Considerations and HPC Transparency

Most high-performance computing (HPC) centers require full transparency regarding the simulations performed on their infrastructure. For this reason, **encryption support is an optional compile-time feature** controlled by the FOAM_USE_SODIUM flag. Centers may choose not to enable this functionality.

It is important to understand what encryption does and does not protect:

- **What is protected:** The time-varying boundary field data in constant/boundaryData. When downloaded, this data is sealed and cannot be read without the private key.
- **What is NOT protected:** The test case setup (mesh, dictionaries, initial conditions) remains unencrypted. Anyone can re-run the global simulation to regenerate the boundary data—encryption does not prevent this.

The practical security model relies on the following observations:

1. Re-running the global simulation to obtain internal fields is **computationally expensive** and requires HPC resources. Such jobs are recorded in the scheduler logs, providing an audit trail.
2. Offline recalculation on a personal workstation is typically infeasible due to computational cost.
3. If file permissions on HPC scratch directories are misconfigured and the user chooses not to write timestep data from the global simulation, other users can only copy the unencrypted test case setup—not the valuable boundary data.

i Note

The encryption feature is designed for protecting intellectual property during data distribution, not for hiding simulations from HPC administrators. Always comply with your computing center's acceptable use policies.

3.7 Protection Against Malware and Ransomware

The spaceTimeWindow approach offers an additional security benefit: by storing only the minimal data required for reconstruction, the valuable simulation results can be protected against malware and ransomware attacks.

3.7.1 Secure Archival Strategy

The reconstruction case contains only:

- The subset mesh (`constant/polyMesh`)
- Encrypted boundary data (`constant/boundaryData/*.zstd.enc` or `*.enc`)
- Initial fields for one timestep
- Configuration dictionaries

This minimal dataset (typically a few gigabytes) can be stored on:

- **Write-Once Read-Many (WORM) media:** Optical discs (M-DISC), tape archives with WORM capability, or cloud storage with object lock/immutability settings (AWS S3 Object Lock, Azure Immutable Blob Storage).
- **Secure vaults:** Hardware-encrypted drives kept offline, or institutional secure storage with access controls and audit logging.
- **Air-gapped backups:** Disconnected storage that cannot be reached by network-based malware.

3.7.2 Security Model

1. **Encrypted boundary data is immutable:** Once written to WORM storage, the encrypted files cannot be modified or deleted by malware.
2. **Private key stored separately:** The decryption key resides on a hardware token or secure password manager, not on the same system as the data.
3. **Reconstruction is always possible:** Even if the HPC system or workstation is compromised, the archived data remains intact and can be decrypted on a clean system.
4. **Minimal attack surface:** The small archive size makes backup and verification practical, unlike terabyte-scale full simulation outputs.

✓ Tip

For critical simulations, archive the encrypted reconstruction case to WORM storage immediately after extraction completes. The full simulation data can then be deleted from HPC scratch space, eliminating the risk of data loss from ransomware or accidental deletion.

4 Boundary Condition Approaches

The library provides five approaches for applying boundary conditions on the extraction boundary (`oldInternalFaces`), plus an optional ghost cell mode that can be combined with any approach:

4.1 Pressure-Coupled BC — Recommended

The **recommended approach** for accurate reconstruction uses `spaceTimeWindowCoupledPressure` for the pressure field, which properly handles its elliptic nature.

- **Velocity (U):** `spaceTimeWindow` — pure Dirichlet from extracted data
- **Pressure (p):** `spaceTimeWindowCoupledPressure` — mixed Dirichlet/Neumann based on flux
- **Turbulence fields:** `zeroGradient`

Why this approach is physically correct:

- Pressure is governed by an elliptic Poisson equation — cutting the domain severs the pressure’s ability to “feel” the flow outside the window
- The BC uses smooth tanh-based blending to transition between Dirichlet and Neumann limits
- Two blending modes are available:
 - `fluxMagnitude` (default, recommended): Dirichlet at stagnant faces, Neumann at active flow faces — more compatible with continuity
 - `flowDirection`: Dirichlet at inflow faces, Neumann at outflow faces — the original approach
- This **reproduces the original flow** rather than computing a different solution

```

1 // In 0/U
2 oldInternalFaces
3 {
4     type            spaceTimeWindow;
5     dataDir         "constant/boundaryData";
6     allowTimeInterpolation true;
7     timeInterpolationScheme cubic;
8     value           uniform (0 0 0);
9 }
10
11 // In 0/p
12 oldInternalFaces
13 {
14     type            spaceTimeWindowCoupledPressure;
15     dataDir         "constant/boundaryData";
16     phi             phi;
17     blendingMode     fluxMagnitude; // or flowDirection
18     dirichletWeight  0.8;           // Dirichlet strength (0-1)
19     neumannWeight    0.8;           // Neumann strength (0-1)
20     transitionWidth  0.1;           // Blending sharpness
21     allowTimeInterpolation true;
22     timeInterpolationScheme cubic;
23     value           uniform 0;

```

24 }

Listing 6: Pressure-coupled boundary conditions**i Note**

The pressure-coupled approach requires extracting pressure gradients during extraction. Enable this with `extractGradients true` and `gradientFields (p)` in the extraction configuration. See section 6.1 for details.

4.2 Precise BC (-preciseBC) – Highest Fidelity Reconstruction

The `-preciseBC` approach is the most "accurate" reconstruction option, using the intermediate velocity $\mathbf{u}^* = H(\mathbf{U})/A$ (commonly written `HbyA` in OpenFOAM) instead of the final velocity \mathbf{U} as the boundary condition for the velocity field.

In OpenFOAM this involves modifying the solver to expose `HbyA` to function objects.

- **Velocity (U):** `spaceTimeWindow` with `fieldTableName HbyA` — reads extracted \mathbf{u}^* data
- **Pressure (p):** `spaceTimeWindowCoupledPressure` — mixed Dirichlet/Neumann based on flux
- **Turbulence fields:** `zeroGradient`

Why \mathbf{u}^* is the theoretically optimal boundary condition:

In the PIMPLE algorithm, the pressure Poisson equation is:

$$\nabla \cdot \left(\frac{1}{A} \nabla p \right) = \nabla \cdot \mathbf{u}^*$$

The intermediate velocity \mathbf{u}^* is the *source term* of the pressure equation. If \mathbf{u}^* is prescribed exactly on the boundary, the pressure equation on the subdomain becomes identical to the pressure equation on the full domain (restricted to the subdomain).

Wu et al. [8] in 2020 demonstrated machine-precision pressure recovery using this approach in a DNS context for compressing turbulence databases. Two factors contribute to this: (i) very high spatial resolution minimizes the difference between full-domain and subdomain discretization stencils at boundary cells, and (ii) DNS codes typically employ spectral or direct (non-iterative) pressure solvers that yield machine-precision solutions, unlike the iterative solvers (e.g. GAMG) used in finite-volume codes such as OpenFOAM. On practical LES/RANS meshes with iterative solvers, a measurable reconstruction error remains (see section 4.3), even when:

- Face values use OpenFOAM's exact distance-weighted interpolation formula (no spatial interpolation error), and
- Boundary data is available at every solver timestep (no temporal interpolation error).

The remaining error arises from the inherent difference between solving the non-linear PIMPLE equations on a full domain versus on a subdomain with prescribed boundary conditions [5, 6]. At boundary faces, the discretization changes from an internal-face stencil (using neighbor cell values) to a Dirichlet BC stencil. While the Laplacian gradient is algebraically identical, the non-linear convective terms, TVD/upwind flux limiters, and the multigrid pressure solver all

behave differently at boundary faces [5, 7], introducing small differences that compound over timesteps. It is worth noting that LES and RANS are by definition approximate solutions—the turbulence closure models introduce modelling errors that are typically orders of magnitude larger than the reconstruction error. Nevertheless, -preciseBC remains the most accurate option because it provides the theoretically exact source term for the pressure equation.

In contrast, using the final velocity \mathbf{U} (which satisfies $\nabla \cdot \mathbf{U} = 0$) provides a divergence-free boundary condition that carries no information about the pressure source term, requiring the coupled pressure BC to reconstruct the missing information through blending.

Requirements:

- The extraction must be performed with `pimpleFoamPrecise` (see appendix C), which registers \mathbf{u}^* as `HbyA` in the `objectRegistry`
- The extraction function object must use `fields (HbyA p)` with `extractGradients true` and `gradientFields (p)`

```

1 // In 0/U (reads HbyA data, applied to U field)
2 oldInternalFaces
3 {
4     type                spaceTimeWindow;
5     dataDir              "constant/boundaryData";
6     fieldTableName       HbyA;          // Read HbyA data files for this field
7     fixesValue           true;
8     allowTimeInterpolation true;
9     timeInterpolationScheme cubic;
10    value                 uniform (0 0 0);
11 }
12
13 // In 0/p (same as pressure-coupled)
14 oldInternalFaces
15 {
16     type                spaceTimeWindowCoupledPressure;
17     dataDir              "constant/boundaryData";
18     blendingMode         fluxMagnitude;
19     dirichletWeight       0.8;
20     neumannWeight         0.8;
21     transitionWidth       0.1;
22     allowTimeInterpolation true;
23     timeInterpolationScheme cubic;
24     value                 uniform 0;
25 }

```

Listing 7: Precise boundary conditions (with -preciseBC)

```

1 # Extraction with pimpleFoamPrecise
2 cd source-case
3 pimpleFoamPrecise    # Registers HbyA in objectRegistry
4
5 # Initialization with -preciseBC
6 cd ../subset-case

```

```

7 spaceTimeWindowInitCase -sourceCase ../source-case -preciseBC
8
9 # Reconstruction with standard pimpleFoam
10 pimpleFoam

```

Listing 8: Using precise BC

✓ **Tip**

The `-preciseBC` approach is the highest-fidelity option available. It yields continuity errors $\sim 10^{-19}$ (machine epsilon), compared to the standard pressure-coupled approach which is accurate to solver tolerance ($\sim 10^{-6}$). The remaining reconstruction error arises from the non-linear discretization difference between internal faces (full domain) and Dirichlet boundary faces (subdomain) [5–7], not from any interpolation.

4.3 Accuracy Comparison

A quantitative comparison between `-preciseBC` (using $\mathbf{u}^* + \nabla p$) and the standard pressure-coupled BC (using $\mathbf{U} + \nabla p$) was performed on the UFR2-02 square cylinder case (110 720 cells full domain, 47 952 cells subset, 88 timesteps with $\Delta t = 10^{-4}$, no time interpolation, binary uncompressed data throughout).

Table 1: Internal field errors after 88 timesteps: subset vs. full domain

Method	Velocity \mathbf{U}		Pressure p	
	L_∞	L_2	L_∞	L_2
<code>-preciseBC</code> ($\mathbf{u}^* + \nabla p$)	3.71×10^{-3}	3.07×10^{-4}	2.19×10^{-3}	3.22×10^{-4}
Coupled pressure ($\mathbf{U} + \nabla p$)	3.65×10^{-3}	3.10×10^{-4}	2.65×10^{-3}	3.50×10^{-4}

Both methods start from identical binary initial conditions (zero error at t_0). Key observations:

- **Velocity errors are nearly identical** between the two methods — the choice of \mathbf{u}^* vs. \mathbf{U} for the velocity BC has negligible effect on velocity reconstruction.
- **Pressure errors are ~20% lower with `-preciseBC`:** $L_\infty = 2.19 \times 10^{-3}$ vs. 2.65×10^{-3} .
- **Errors are concentrated near the boundary.** Only 0.25% of cells exceed 90% of the maximum error. The top-10 worst cells are all at $x = 0.055$ (one cell layer from the extraction box face at $x = 0.05$).

In this comparison, no temporal interpolation is used (boundary data is available at every solver timestep) and the face values are computed using OpenFOAM’s exact distance-weighted linear interpolation formula ($\phi_f = w \phi_{\text{inside}} + (1-w) \phi_{\text{outside}}$). Despite this, a reconstruction error remains. It arises from the non-linear discretization difference between full-domain internal faces and subdomain Dirichlet boundary faces [5, 6]: convective flux limiters (TVD/upwind), the multigrid pressure solver (GAMG), and the PIMPLE outer correction loop all behave differently at boundary faces [7], introducing small differences that compound over the 88 timesteps.

Table 2: Error decay with distance from the oldInternalFaces boundary (-preciseBC)

Distance from boundary	Cells	U error		p error	
		Mean	Max	Mean	Max
0.005–0.01 (1 cell layer)	1 296	1.2×10^{-3}	3.7×10^{-3}	1.0×10^{-3}	2.2×10^{-3}
0.01–0.02	5 040	2.5×10^{-4}	9.9×10^{-4}	4.0×10^{-4}	1.5×10^{-3}
0.02–0.05	12 384	1.1×10^{-4}	6.5×10^{-4}	2.4×10^{-4}	8.3×10^{-4}
0.05–0.10	15 156	4.5×10^{-5}	2.9×10^{-4}	1.5×10^{-4}	7.5×10^{-4}
0.10–0.20	14 076	1.2×10^{-5}	9.2×10^{-5}	5.3×10^{-5}	5.5×10^{-4}

i Note

Wu et al. [8] in 2020 report machine-precision pressure recovery in a DNS setting. Two factors explain this: the very high spatial resolution of DNS minimizes the discretization stencil mismatch at boundary cells, and DNS codes typically use spectral or direct (non-iterative) pressure solvers that produce exact solutions. In contrast, finite-volume LES/RANS codes like OpenFOAM use iterative solvers (GAMG) that converge to a tolerance, and coarser grids amplify the stencil mismatch, leading to the measured errors. Moreover, LES and RANS are inherently approximate—turbulence closure models introduce modelling errors that dwarf the reconstruction error. The advantage of \mathbf{u}^* is most visible in the pressure field near the boundary ($\sim 20\%$ lower L_∞).

4.4 Inlet-Outlet BC (-inletOutletBC) — For Quick Tests

Uses spaceTimeWindowInletOutlet BC which applies:

- **Velocity (U):** Flux-based switching — Dirichlet (prescribed value) at inflow faces, zeroGradient at outflow faces
- **All scalar fields (p, nut, k, epsilon, omega):** zeroGradient

This approach is useful for quick tests and situations where exact reproduction of the original flow is not required:

- Turbulent flows have instantaneous velocity fluctuations that can reverse direction locally
- The flux-based switching automatically adapts to the instantaneous flow direction at each face
- However, pressure uses zeroGradient, which can lead to solution drift over time

```
1 spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC
```

Listing 9: Using inlet-outlet BC**⚠ Warning**

The inlet-outlet BC does not reproduce the original pressure field. The subset simulation will evolve to a **different solution** because pressure information from outside the ex-

traction window is lost. For accurate reconstruction, use the pressure-coupled approach instead.

4.5 Fixed Outlet Direction (-outletDirection)

Creates a separate outlet patch from faces at one edge of the extraction box:

- **oldInternalFaces:** Pure Dirichlet BC (spaceTimeWindow) on all remaining faces
- **outlet:** Pressure relief patch with inletOutlet for U and zeroGradient for p

Use when the mean flow direction is well-defined and outflow always occurs at a known location.

```
1 spaceTimeWindowInitCase -sourceCase ../source-case -outletDirection "(1 0 0)"
```

Listing 10: Using fixed outlet direction

△ Warning

The options -preciseBC, -inletOutletBC, and -outletDirection are mutually exclusive. Using incompatible combinations produces an error with guidance.

4.6 Ghost Cell Mode (-ghostCells) — Machine-Precision Reconstruction

The -ghostCells option includes one extra layer of cells (“ghost cells”) outside the extraction boundary in the sub-mesh. Ghost cell values are prescribed from the extracted global simulation data at each timestep using either an fvOption or explicit solver-side overwrite. This pushes the boundary stencil mismatch (see section 2.6) one layer outward, where it only affects ghost cells that are overwritten anyway. The original extraction boundary faces remain **internal** in the sub-mesh, preserving the identical FVM stencil as the global simulation.

Why ghost cells achieve machine-precision accuracy:

- Without ghost cells, extracting a subdomain converts internal faces to boundary faces, changing the FVM discretization stencil—TVD limiters, GAMG coarsening, and gradient reconstruction all behave differently at boundary faces. This limits accuracy to $\sim O(10^{-3})$.
- With ghost cells, the original extraction boundary faces stay internal. The stencil mismatch is pushed to the outer ghost boundary, affecting only ghost cells which are overwritten from extracted data.
- All ghost cell data uses direct cell-centre values (no interpolation), preserving machine-precision transfer at every stage of the pipeline.

Figure 9 illustrates the ghost cell strategy.

Two ghost cell forcing mechanisms are provided:

1. **ghostCellConstraint fvOption** (recommended): Overrides the matrix equation in ghost cells via `eqn.setValues()`. Works with any solver that supports `fvOptions.constrain()`.

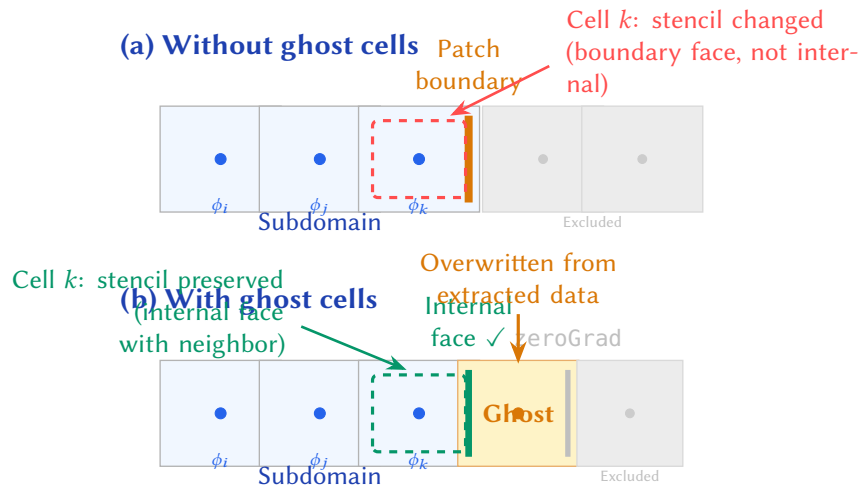


Figure 9: Ghost cell strategy. (a) Without ghost cells, the extraction boundary becomes a patch boundary face, altering the FVM discretization stencil for adjacent cells (different limiters, gradient reconstruction, and GAMG coarsening behaviour). (b) With one layer of ghost cells, the original extraction boundary remains an internal face with the identical stencil as in the global simulation. Ghost cell values are overwritten from extracted data at each timestep; the outer boundary uses zeroGradient (its effect is confined to ghost cells that are overwritten anyway).

2. **Explicit overwrite** (built into `pimpleFoamStag` and `pimpleFoamPrecise`): Directly writes ghost cell values into the field arrays. Activated automatically when a `ghostCells` `cellZone` exists in the mesh.

```

1  extractSubset
2  {
3      type          spaceTimeWindowExtract;
4      libs          (spaceTimeWindow);
5      box           ((0.05 -0.15 0.01) (0.70 0.15 0.38));
6      outputDir     "../subset";
7
8      fields        (U p);
9      initialFields (U p nut);
10
11     extractGhostCells true;          // Enable ghost cell data extraction
12
13     writeFormat    deltaVarint;
14     writeControl    timeStep;
15     writeInterval   1;
16 }

```

Listing 11: Extraction configuration with ghost cells

```

1  # Case initialization with ghost cells
2  spaceTimeWindowInitCase -sourceCase ../source-case -ghostCells
3

```

```

4 # Run reconstruction (ghost cell overwrite is automatic)
5 pimpleFoamStag # or pimpleFoamPrecise

```

Listing 12: Ghost cell reconstruction workflow

```

1 // In constant/fvOptions or system/fvSolution
2 ghostU
3 {
4     type            vectorGhostCellConstraint;
5     selectionMode    cellZone;
6     cellZone         ghostCells;
7     dataDir          "constant/boundaryData";
8     fields           (U);
9 }
10
11 ghostP
12 {
13     type            scalarGhostCellConstraint;
14     selectionMode    cellZone;
15     cellZone         ghostCells;
16     dataDir          "constant/boundaryData";
17     fields           (p);
18 }

```

Listing 13: ghostCellConstraint fvOption configuration

i Note

When `-ghostCells` is used, the `oldInternalFaces` patch moves one layer outward (to the ghost cell outer boundary). All fields on this patch receive `zeroGradient` BCs by default, since the ghost cells are overwritten from extracted data and the outer boundary condition does not affect the interior solution.

i Note

Ghost cell mode can be combined with any BC approach. For example, `-ghostCells` with `-preciseBC` uses ghost cell forcing for the interior stencil and u^* data on the (now-internal) original extraction boundary.

4.7 Boundary Condition Assignment Logic

For each field in the initial time directory, `spaceTimeWindowInitCase` assigns BCs according to Table 3.

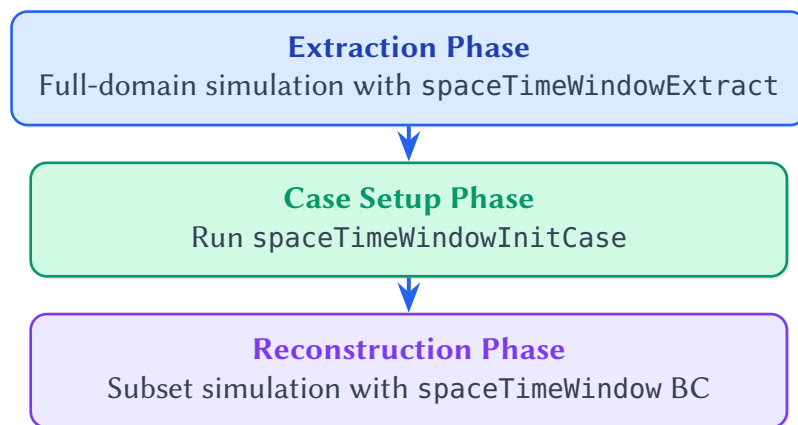
This ensures fields without time-varying data automatically get appropriate BCs. With `-preciseBC`, the `U` and `p` fields are handled specially regardless of `boundaryData` contents (`U` reads `HbyA` data via `fieldTableName`).

Table 3: Boundary condition assignment logic

Mode	Field	In BD?	Type	BC Applied
-preciseBC	U	—	vector	spaceTimeWindow + fieldTableName HbyA
-preciseBC	p	—	scalar	spaceTimeWindowCoupledPressure
-preciseBC	other	No	any	zeroGradient
-inletOutletBC	—	Yes	vector	spaceTimeWindowInletOutlet
-inletOutletBC	—	Yes	scalar	zeroGradient
default	—	Yes	any	spaceTimeWindow (Dirichlet)
any	—	No	any	zeroGradient

5 Workflow Overview

The space-time window reconstruction workflow is strictly sequential, as shown in fig. 10.

**Figure 10:** Space-time window reconstruction workflow

5.1 Serial Workflow

```

1 # 1. Run extraction during simulation
2 cd source-case
3 pimpleFoam      # With spaceTimeWindowExtract function object
4
5 # 2. Initialize reconstruction case (recommended: inlet-outlet BC)
6 cd ../subset-case
7 spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC
8
9 # 3. Run reconstruction
10 pimpleFoam
  
```

Listing 14: Complete serial workflow

In serial mode:

- Mesh and initial fields are written directly to the output directory
- Boundary data is written at every timestep

- The extractor forces field writes at t_1 and t_2 (for interpolation buffers)

i Note

In serial mode, the extractor writes internal fields only at t_0 (extraction start). The t_1 and t_2 internal fields are **not** automatically saved. If you need linear interpolation starting at t_1 , ensure your `writeInterval` captures t_1 , or use parallel mode which forces writes at t_0 , t_1 , and t_2 .

5.2 Parallel Workflow

1. Run the original simulation in parallel with extraction enabled
2. Run `reconstructPar` for the desired start timestep (t_2 for cubic interpolation)
3. Run `spaceTimeWindowInitCase -sourceCase <path> -inletOutletBC`
4. Run the solver on the subset case (serial or parallel)

```

1 # Step 1: Run parallel extraction
2 cd source-case
3 mpirun -np 4 pimpleFoam -parallel
4 # Extraction automatically forces writes at t_0, t_1, t_2
5
6 # Step 2: Reconstruct fields at cubic-safe start time (t_2)
7 reconstructPar -time 0.0002
8
9 # Step 3: Initialize subset case (recommended: inlet-outlet BC)
10 cd ../subset-case
11 spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC
12
13 # Step 4: Run reconstruction
14 pimpleFoam

```

Listing 15: Complete parallel workflow

In parallel mode:

- Boundary data from all processors is gathered to master and written as single files
- The extractor forces field writes at t_0 , t_1 , and t_2 for interpolation buffers (t_0 only in parallel)
- Only extraction box parameters are written (not mesh) — `spaceTimeWindowInitCase` creates the mesh
- `reconstructPar` must be run at the reconstruction start time (t_2) to provide source fields

6 Configuration Reference

6.1 Extraction Configuration

The extraction function object is configured in `system/controlDict` within the `functions` sub-dictionary. The available parameters are listed in table 4.

```

1  functions
2  {
3      extractSubset
4      {
5          type          spaceTimeWindowExtract;
6          libs          (spaceTimeWindow);
7
8          // Bounding box: ((minX minY minZ) (maxX maxY maxZ))
9          // Must be fully internal to domain
10         box           ((0.05 -0.25 0.01) (0.90 0.25 0.38));
11
12         outputDir      "../subset-case";
13
14         // Fields for time-varying boundary data (written every timestep)
15         fields         (U p);           // U and p for pressure-coupled BC
16
17         // Fields for initial conditions (optional, defaults to 'fields')
18         initialFields  (U p nut);       // nut needed for turbulence model
19
20         // Gradient extraction for pressure-coupled BC (RECOMMENDED)
21         extractGradients true;
22         gradientFields (p);           // Extract normal gradient of p
23
24         // Write format: ascii, binary, deltaVarint, or dvzt
25         writeFormat    dvzt;          // Recommended: best compression
26
27         // Precision for deltaVarint/dvzt (default: 6)
28         deltaVarintPrecision 6;
29
30         // Keyframe interval for dvzt (default: 20)
31         dvztKeyframeInterval 20;
32
33         // Gzip compression (ignored for deltaVarint/dvzt)
34         writeCompression off;
35
36         // Ghost cell extraction (for machine-precision reconstruction)
37         // extractGhostCells true;
38
39         // Optional encryption
40         // publicKey     "base64-encoded-public-key";
41
42         // Time window (optional)
43         // timeStart     0.0;
44         // timeEnd       1.0;
45
46         writeControl    timeStep;
47         writeInterval   1;
48     }
49 }

```

Listing 16: Complete extraction configuration

Table 4: Extraction parameters

Parameter	Type	Required	Description
box	pointPair	Yes	Extraction bounding box
outputDir	fileName	Yes	Output case directory
fields	wordList	Yes	Fields for time-varying BC (boundary-Data)
initialFields	wordList	No	Fields for initial conditions (default: same as fields)
extractGradients	bool	No	Enable gradient extraction (default: false)
gradientFields	wordList	No	Fields for gradient extraction (requires extractGradients)
extractGhostCells	bool	No	Extract ghost cell data for machine-precision reconstruction (default: false)
writeFormat	word	No	ascii, binary, deltaVarint, or dvzt
deltaVarintPrecision	label	No	Decimal digits (default: 6)
dvztKeyframeInterval	label	No	Keyframe interval for dvzt (default: 20)
writeCompression	switch	No	Gzip compression (ignored for deltaVarint/dvzt)
publicKey	string	No	Base64 public key for encryption
timeStart	scalar	No	Extraction start time
timeEnd	scalar	No	Extraction end time

6.2 Initial Fields vs Boundary Data Fields

The extraction can separate which fields are used for:

- **Initial conditions** (initialFields): Fields written to the start time directory for solver initialization
- **Time-varying boundary data** (fields): Fields written to boundaryData/ for time-varying BCs

For the highest-fidelity approach with -preciseBC (requires pimpleFoamPrecise):

```

1 // In extraction function object (run with pimpleFoamPrecise)
2 fields          (HbyA p);          // HbyA (intermediate velocity) and p for
   time-varying BC
3 initialFields   (U p nut);          // U (final velocity) needed for solver
   initialization
4 extractGradients true;              // Enable gradient extraction
5 gradientFields  (p);                // Extract normal gradient of p (creates
   gradp)

```

Listing 17: Field selection for precise BC (highest fidelity)

For the standard pressure-coupled approach:

```

1 // In extraction function object (run with pimpleFoam)
2 fields          (U p);              // U and p for time-varying BC

```

```

3  initialFields    (U p nut);           // nut needed for turbulence model
4  extractGradients true;               // Enable gradient extraction
5  gradientFields   (p);                 // Extract normal gradient of p (creates
    gradp)

```

Listing 18: Field selection strategy for pressure-coupled BC

Note: nut is only needed in initialFields (not fields) since the turbulence model computes it at runtime and uses zeroGradient BC on oldInternalFaces.

Fields NOT in boundaryData automatically get zeroGradient BC on oldInternalFaces.

✓ Tip

For accurate reconstruction, use the **pressure-coupled BC approach**: extract both U and p with gradient extraction enabled for p. This allows the spaceTimeWindowCoupledPressure BC to properly constrain the pressure field and reproduce the original solution.

i Note

When extractGradients is enabled, the extractor computes the face-normal gradient at each boundary face using the equation:

$$\left. \frac{\partial \phi}{\partial n} \right|_f = \frac{\phi_N - \phi_P}{|\mathbf{d}_{PN}|}$$

where ϕ_P is the value at the face owner cell, ϕ_N is the value at the neighbour cell, and \mathbf{d}_{PN} is the vector from owner to neighbour cell centre. The gradient data is written to files named grad<fieldName> (e.g., gradp) in the boundaryData directory.

6.3 Boundary Condition Configuration

6.3.1 spaceTimeWindowCoupledPressure (Recommended for Pressure)

Mixed (Robin) boundary condition for pressure that blends Dirichlet and Neumann conditions based on local flux. This is the **recommended approach** for pressure at extraction boundaries. Two blending modes are available:

- **fluxMagnitude** (default, recommended): Dirichlet at stagnant faces (zero flux), Neumann at active flow faces — better continuity compatibility
- **flowDirection**: Dirichlet at inflow, Neumann at outflow — the original flow-direction-based approach

The parameters are listed in table 5.

```

1  boundaryField
2  {
3      oldInternalFaces
4      {
5          type                spaceTimeWindowCoupledPressure;
6          dataDir              "constant/boundaryData";
7          phi                  phi;           // Flux field name

```

```

8      blendingMode          fluxMagnitude; // or flowDirection
9      dirichletWeight       0.8;          // Dirichlet strength (0-1)
10     neumannWeight         0.8;          // Neumann strength (0-1)
11     transitionWidth       0.1;          // Blending sharpness
12     allowTimeInterpolation true;
13     timeInterpolationScheme cubic;      // or "linear" or "none"
14     value                  uniform 0;
15 }
16 }

```

Listing 19: spaceTimeWindowCoupledPressure configuration

How it works:

1. Reads pressure values and pressure gradients from boundaryData
2. Computes local flux at each face
3. Calculates blending weight using smooth tanh transition based on the selected mode:

- fluxMagnitude:

$$w = w_D + (w'_N - w_D) \cdot \tanh(|\phi_f|/\phi_{\max}/\sigma)$$

where w_D is dirichletWeight and $w'_N = 1 - w_N$

- flowDirection:

$$w = \frac{1}{2} \left[(w_D + w'_N) - (w_D - w'_N) \cdot \tanh(\phi_f/\phi_{\max}/\sigma) \right]$$

4. In fluxMagnitude mode: at **stagnant faces** ($w \rightarrow w_D$) applies Dirichlet; at **high-flux faces** ($w \rightarrow w'_N$) applies Neumann
5. In flowDirection mode: at **inflow faces** ($w \rightarrow w_D$) applies Dirichlet; at **outflow faces** ($w \rightarrow w'_N$) applies Neumann

The value coefficients are:

$$\text{valueFraction} = w \cdot w_D + (1 - w) \cdot 0 = w \cdot w_D \quad (6)$$

$$\text{refValue} = p_{\text{extracted}} \quad (7)$$

$$\text{refGrad} = \left. \frac{\partial p}{\partial n} \right|_{\text{extracted}} \quad (8)$$

where w_D is dirichletWeight and the gradient coefficient is $(1 - w) \cdot w_N$ with w_N being neumannWeight.

i Note

This BC requires gradient data in boundaryData. Enable gradient extraction with `extractGradients true` and `gradientFields (p)` in the extraction configuration. The gradient file must be named `gradp` in each timestep directory.

Table 5: spaceTimeWindowCoupledPressure parameters

Parameter	Type	Default	Description
dataDir	fileName	constant/boundaryData	Path to boundary data
phi	word	phi	Name of flux field
blendingMode	word	fluxMagnitude	fluxMagnitude or flowDirection
dirichletWeight	scalar	0.8	Dirichlet contribution strength (0-1)
neumannWeight	scalar	0.8	Neumann contribution strength (0-1)
transitionWidth	scalar	0.1	Sharpness of flux transition
allowTimeInterpolation	bool	false	Permit interpolation for missing timesteps
timeInterpolationScheme	word	linear	none, linear, or cubic

6.3.2 spaceTimeWindowInletOutlet

Flux-based boundary condition that reads pre-computed velocity values and applies them only at inflow faces. The parameters are listed in table 6.

```

1 boundaryField
2 {
3     oldInternalFaces
4     {
5         type                spaceTimeWindowInletOutlet;
6         dataDir              "constant/boundaryData";
7         phi                  phi;           // Flux field name
8         allowTimeInterpolation true;
9         timeInterpolationScheme cubic;      // or "linear" or "none"
10        value                uniform (0 0 0);
11    }
12 }
```

Listing 20: spaceTimeWindowInletOutlet configuration

How it works:

1. At each timestep, reads velocity values from boundaryData (with optional time interpolation)
2. Computes flux through each face: $\phi_f = \mathbf{U}_f \cdot \mathbf{S}_f$
3. For inflow faces ($\phi < 0$): applies prescribed velocity from boundaryData
4. For outflow faces ($\phi \geq 0$): applies zeroGradient (extrapolates from interior)

Table 6: spaceTimeWindowInletOutlet parameters

Parameter	Type	Default	Description
dataDir	fileName	constant/boundaryData	Path to boundary data
phi	word	phi	Name of flux field
allowTimeInterpolation	bool	false	Permit interpolation for missing timesteps
timeInterpolationScheme	word	linear	none, linear, or cubic

6.3.3 spaceTimeWindow (Pure Dirichlet)

Pure Dirichlet boundary condition that prescribes values on all faces regardless of flow direction. The parameters are listed in table 7.

```

1 boundaryField
2 {
3     oldInternalFaces
4     {
5         type                spaceTimeWindow;
6         dataDir              "constant/boundaryData";
7         fixesValue           true;           // Tells adjustPhi these values
            are fixed
8         allowTimeInterpolation true;
9         timeInterpolationScheme cubic;
10        reportFlux           true;
11        value                 uniform (0 0 0);
12    }
13 }
```

Listing 21: spaceTimeWindow configuration

Use this for:

- Scalar fields when not using -inletOutletBC
- Situations where fixed values are explicitly desired on all faces

6.4 Case Initialization Options

The spaceTimeWindowInitCase utility accepts command-line options (see table 8):

```

1 # Highest fidelity: precise BC (requires pimpleFoamPrecise extraction)
2 spaceTimeWindowInitCase -sourceCase ../source-case -preciseBC
3
4 # Standard: pressure-coupled BC (default, no flag needed)
5 spaceTimeWindowInitCase -sourceCase ../source-case
6
7 # Quick tests: inlet-outlet BC
8 spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC
9
```

Table 7: spaceTimeWindow parameters

Parameter	Type	Default	Description
dataDir	fileName	constant/boundaryData	Path to boundary data
fixesValue	bool	true	Report to adjustPhi that values are fixed
allowTimeInterpolation	bool	false	Enable time interpolation
timeInterpolationScheme	word	linear	none, linear, or cubic
reportFlux	bool	false	Print net flux through patch (velocity only)
setAverage	bool	false	Adjust field average
offset	Type	Zero	Offset value

```

10 # Alternative: fixed outlet direction for steady-mean flows
11 spaceTimeWindowInitCase -sourceCase ../source-case -outletDirection "(1 0 0)"
12
13 # Ghost cell mode: machine-precision reconstruction
14 spaceTimeWindowInitCase -sourceCase ../source-case -ghostCells
15
16 # With mass flux correction (optional, ensures exact mass conservation)
17 spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC
    -correctMassFlux

```

Listing 22: spaceTimeWindowInitCase usage**Table 8:** spaceTimeWindowInitCase options

Option	Type	Description
-sourceCase	directory	Source case where extraction ran (required)
-extractDir	directory	Directory with extracted data (default: cwd)
-preciseBC	flag	Use HbyA-based velocity BC and coupledPressure for p (requires pimpleFoamPrecise extraction)
-inletOutletBC	flag	Use flux-based inlet-outlet BC for U, zeroGradient for scalars (for quick tests)
-outletDirection	vector	Create fixed outlet patch in given direction (e.g., "(1 0 0)")
-outletFraction	scalar	Fraction of box extent for outlet region (default: 0.1)
-ghostCells	flag	Include one layer of ghost cells outside extraction boundary for machine-precision reconstruction (see section 4.6)
-correctMassFlux	flag	Apply least-squares mass flux correction to boundaryData
-initialFields	list	Override initial fields list (e.g., "(U p nut k)")
-refineLevel	label	Refine mesh N times (spatial interpolation at runtime)
-coarsenLevel	label	Coarsen mesh N times (spatial interpolation at runtime)
-overwrite	flag	Overwrite existing files

△ Warning

`-preciseBC`, `-inletOutletBC`, and `-outletDirection` are mutually exclusive. Using incompatible combinations produces an error with guidance on which option to choose.

6.5 Mesh Coarsening and Refinement

The reconstruction mesh can be coarsened or refined relative to the extraction mesh (table 9). This enables running reconstructions at different resolutions than the original simulation.

```

1 # Refine mesh (finer than extraction)
2 spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -refineLevel 1
3
4 # Coarsen mesh (coarser than extraction)
5 spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -coarsenLevel 1
6
7 # Multiple levels
8 spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -refineLevel 2

```

Listing 23: Mesh resolution options

Table 9: Mesh resolution options

Option	Type	Description
<code>-refineLevel</code>	label	Refine mesh N times (each level splits cells $\times 8$)
<code>-coarsenLevel</code>	label	Coarsen mesh N times (each level merges cells)

△ Warning

`-refineLevel` and `-coarsenLevel` are mutually exclusive. Use only one at a time.

6.5.1 Spatial Interpolation Algorithms

When the reconstruction mesh differs from the extraction mesh, spatial interpolation is required for boundary data. The `spaceTimeWindow` boundary conditions handle this automatically at runtime using different algorithms depending on the resolution change.

Refinement: Barycentric Interpolation with 2D Delaunay Triangulation When the target mesh has more faces than the source (refinement), the algorithm triangulates source face centers using the Bowyer-Watson algorithm, as illustrated in fig. 11. For each target face center, the enclosing triangle is found and barycentric weights are computed. If the target point lies outside all triangles (which can happen because the extracted submesh uses original cells from the source case and may have irregular boundaries), the algorithm finds the nearest triangle by centroid distance and uses clamped barycentric coordinates. This provides smooth C^0 continuous interpolation.

Given a target point \mathbf{p} inside a triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, the barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$ satisfy eq. (9):

$$\mathbf{p} = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \lambda_3 \mathbf{v}_3, \quad \text{where} \quad \lambda_1 + \lambda_2 + \lambda_3 = 1 \quad (9)$$

The weights are computed from signed triangle areas (eq. (10)):

$$\lambda_1 = \frac{A(\mathbf{p}, \mathbf{v}_2, \mathbf{v}_3)}{A(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)}, \quad \lambda_2 = \frac{A(\mathbf{v}_1, \mathbf{p}, \mathbf{v}_3)}{A(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)}, \quad \lambda_3 = \frac{A(\mathbf{v}_1, \mathbf{v}_2, \mathbf{p})}{A(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)} \quad (10)$$

where the signed area of a 2D triangle is given by eq. (11):

$$A(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{1}{2} [(b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)] \quad (11)$$

The interpolated field value at the target point is then (eq. (12)):

$$\phi(\mathbf{p}) = \lambda_1 \phi_1 + \lambda_2 \phi_2 + \lambda_3 \phi_3 \quad (12)$$

Barycentric Interpolation (Refinement)

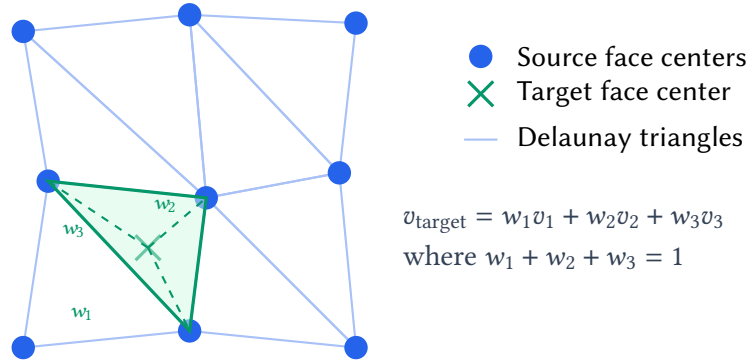


Figure 11: Barycentric interpolation for mesh refinement: source face centers (blue) are triangulated, and each target point (green) is interpolated using barycentric weights from the enclosing triangle

Coarsening: Area-Weighted Averaging When the target mesh has fewer faces than the source (coarsening), an octree is built from source points for efficient spatial lookup, as shown in fig. 12. For each target face, all source points within a search radius are found and averaged with equal weights. If no points are found, the search radius is progressively expanded. This ensures conservation of integral quantities.

For a target point \mathbf{p} with search radius r , define the set of contributing source points (eq. (13)):

$$\mathcal{S}(\mathbf{p}, r) = \{i : \|\mathbf{x}_i - \mathbf{p}\| \leq r\} \quad (13)$$

The interpolated value is the arithmetic mean of all contributing sources (eq. (14)):

$$\phi(\mathbf{p}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \phi_i \quad (14)$$

If $|\mathcal{S}| = 0$, the search radius is expanded by a factor $\alpha > 1$ (typically $\alpha = 1.5$) and the search is repeated (eq. (15)):

$$r \leftarrow \alpha \cdot r \quad \text{until} \quad |\mathcal{S}(\mathbf{p}, r)| > 0 \quad (15)$$

The octree provides $O(\log N)$ lookup complexity for finding points within the search radius, making the algorithm efficient even for large meshes.

Area-Weighted Averaging (Coarsening)

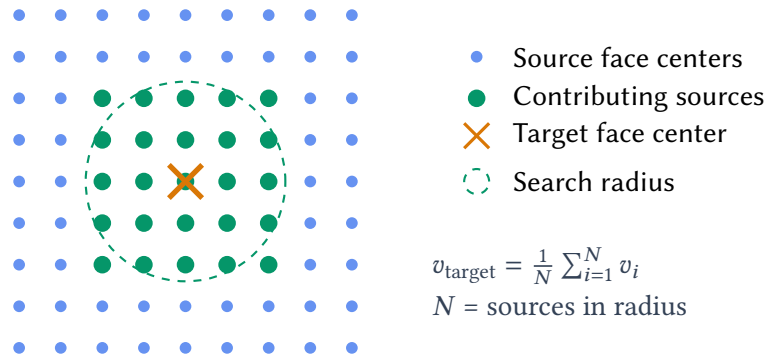


Figure 12: Area-weighted averaging for mesh coarsening: all source face centers (blue) within the search radius of each target point (orange) are averaged with equal weights

2D Projection by Box Face The interpolation is performed on 2D point clouds, as illustrated in fig. 13. Points are grouped by which face of the extraction bounding box they belong to (6 planar surfaces: $\pm X$, $\pm Y$, $\pm Z$), then projected to 2D by dropping the constant coordinate. This exploits the box geometry while handling the irregular face distribution that arises from extracting a submesh with original cell shapes.

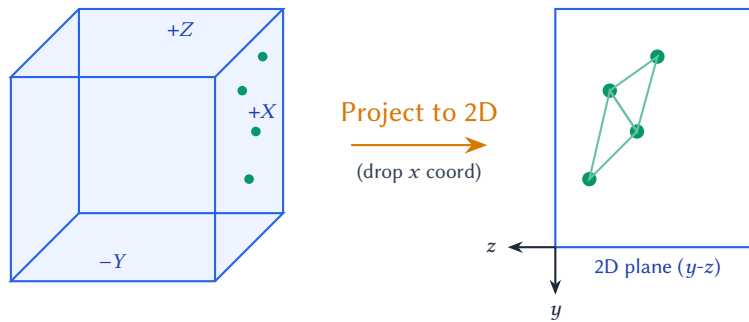


Figure 13: Points on each box face are projected to 2D for triangulation. The constant coordinate (perpendicular to the face) is dropped, enabling efficient 2D algorithms

6.5.2 Initial Field Interpolation

Initial fields are also interpolated when mesh resolution changes (see figs. 14 and 15):

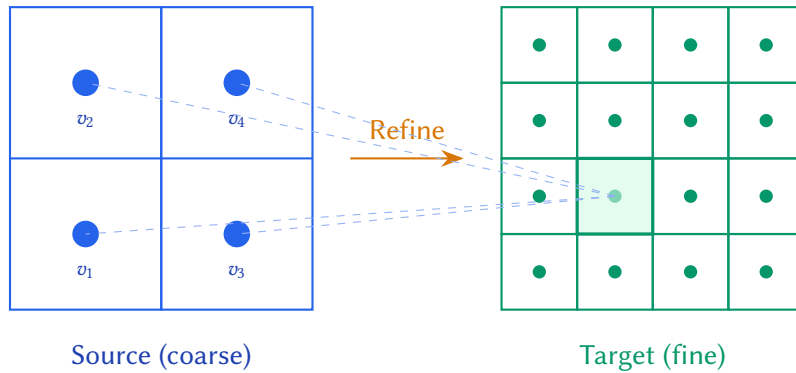
- **Refinement:** Uses `mapFields` with cell-center interpolation (fig. 14)
- **Coarsening:** Uses volume-weighted averaging of source cells (fig. 15)

i Note

Spatial interpolation introduces smoothing, particularly for coarsening. For turbulent flows, this may affect small-scale structures. Consider the trade-off between computational cost and resolution fidelity.

What `spaceTimeWindowInitCase` creates:

mapFields Cell-Center Interpolation (Refinement)



Each target cell interpolates from neighboring source cells

Figure 14: Initial field refinement using mapFields: target cell values are interpolated from surrounding source cell centers using distance-weighted averaging

Volume-Weighted Averaging (Coarsening)

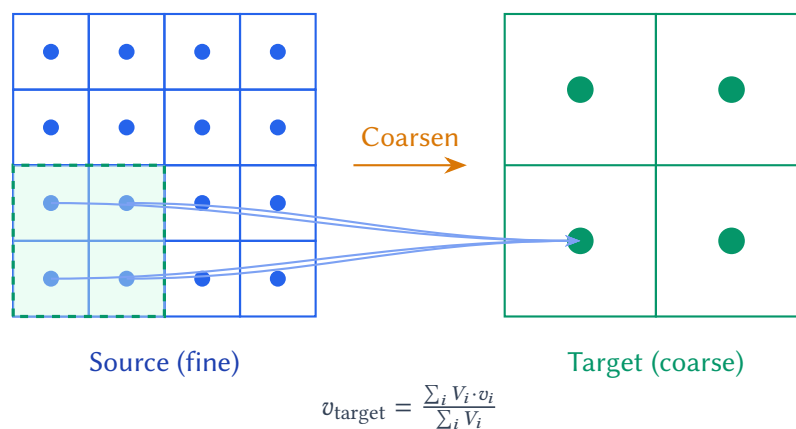


Figure 15: Initial field coarsening using volume-weighted averaging: source cell values within each target cell region are averaged weighted by their volumes

1. `system/controlDict` — With matching solver, `deltaT`, `adjustTimeStep` from extraction
 - `startTime` set to t_2 (third timestep) for cubic interpolation buffer
 - `endTime` set to t_{n-2} (third-to-last) for cubic interpolation buffer
2. `system/fvSchemes`, `system/fvSolution` — Copied from source case (with `pRefPoint` added)
3. `constant/` files — All physics properties copied (mandatory for fidelity)
4. Initial field files with appropriate BCs based on options and `boundaryData` availability

7 Data Storage Format

7.1 Directory Structure

The extraction creates the following directory structure:

```

1  outputDir/
2      constant/
3          polyMesh/                # Subset mesh
4              points
5              faces
6              owner
7              neighbour
8              boundary
9              extractionBox        # (parallel only)
10         boundaryData/
11             oldInternalFaces/
12                 points            # Face centres
13                 extractionMetadata # Settings and timestep list
14                 0.0001/
15                     U              # or U.dvz, U.dvz.zstd, U.dvz.zstd.enc
16                     p              # pressure (for coupled BC)
17                     gradp          # pressure gradient (for coupled BC)
18                 0.0002/
19                 ...
20             ghostCells/           # Only when extractGhostCells is true
21                 cellCentres       # Ghost cell positions (written once)
22                 ghostCellMap      # Data index to subset cell index mapping
23                 0.0001/
24                     U              # Cell-centre values per timestep
25                     p
26                 0.0002/
27                 ...
28         0.0001/                   # Initial fields (serial only)
29             U
30             p
31             nut

```

Listing 24: Extracted data structure

7.2 Boundary Data Compression

The `writeFormat` parameter controls how boundary data files are written, with compression ratios summarized in table 10.

Table 10: Compression comparison

Format	Extension	Typical Size	Notes
ASCII	(none)	100%	Human-readable
Binary	(none)	~50%	OpenFOAM native
ASCII + gzip	.gz	~10%	Compressed
Binary + gzip	.gz	~8%	Compressed
deltaVarint	.dvz	~2.7%	High compression, self-contained
deltaVarint + zstd	.dvz.zstd	~0.4%	Integrated zstd (auto if libzstd)
dvzt	.dvzt	~2.4%	Best compression, recommended
dvzt + zstd	.dvzt.zstd	~0.3%	Best compression + integrated zstd

7.2.1 Delta-Varint Codec (DVZ)

Specialized codec optimized for CFD boundary data (see figs. 16 and 17):

1. **Component-major ordering:** Groups similar values (all U_x , then U_y , then U_z)
2. **Spatial delta encoding:** Stores differences between consecutive face values within the same timestep
3. **Quantization:** Rounds to configurable precision
4. **Varint encoding:** Variable-length integer encoding
5. **Zigzag encoding:** Efficient signed integer representation

i Note

Each DVZ timestep file is completely self-contained. Delta encoding is purely spatial (between consecutive faces in a single field), not temporal. Any timestep can be read independently without access to other timesteps.

7.2.2 DVZ Mathematical Formulation

Component-Major Reordering For a vector field U with N faces, the raw data layout is face-major (eq. (16)):

$$\text{Raw} : [U_{x,0}, U_{y,0}, U_{z,0}, U_{x,1}, U_{y,1}, U_{z,1}, \dots, U_{x,N-1}, U_{y,N-1}, U_{z,N-1}] \quad (16)$$

DVZ reorders to component-major for better compression (eq. (17)):

$$\text{Reordered} : [\underbrace{U_{x,0}, U_{x,1}, \dots, U_{x,N-1}}_{\text{all } x}, \underbrace{U_{y,0}, \dots, U_{y,N-1}}_{\text{all } y}, \underbrace{U_{z,0}, \dots, U_{z,N-1}}_{\text{all } z}] \quad (17)$$

DVZ Encoding Pipeline

1. Raw Data (face-major):

$U_{x,0}$	$U_{y,0}$	$U_{z,0}$	$U_{x,1}$	$U_{y,1}$	$U_{z,1}$	$U_{x,2}$	$U_{y,2}$	$U_{z,2}$
-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------



2. Component-major reorder:

$U_{x,0}$	$U_{x,1}$	$U_{x,2}$	$U_{y,0}$	$U_{y,1}$	$U_{y,2}$	$U_{z,0}$	$U_{z,1}$	$U_{z,2}$
all x			all y			all z		



3. Spatial delta encoding:

v_0	δ_1	δ_2	v_0	δ_1	δ_2	v_0	δ_1	δ_2
-------	------------	------------	-------	------------	------------	-------	------------	------------

$$\delta_i = v_i - v_{i-1}$$



4. Quantize → Zigzag → Varint:

Compact variable-length bytes

Small δ = fewer bytes

Figure 16: DVZ encoding pipeline: data is reordered by component for better locality, then delta-encoded spatially. Quantization, zigzag encoding (for signed integers), and variable-length integer encoding produce compact output

Spatial Delta Encoding

Original values:

1.234	1.238	1.241	1.237	1.240
face 0	face 1	face 2	face 3	face 4

$$\delta_i = v_i - v_{i-1}$$

Deltas are small
 \Rightarrow fewer bits
 \Rightarrow high compression

Delta encoded:

1.234	+0.004	+0.003	-0.004	+0.003
base	delta	delta	delta	delta

Figure 17: Spatial delta encoding: instead of storing absolute values, DVZ stores the first value and differences between consecutive faces. Smooth CFD fields have small deltas, enabling high compression

Spatial Delta Encoding For each component sequence $\{v_0, v_1, \dots, v_{N-1}\}$, store deltas as shown in eq. (18):

$$\delta_0 = v_0, \quad \delta_i = v_i - v_{i-1} \quad \text{for } i = 1, \dots, N-1 \quad (18)$$

For smooth CFD fields, consecutive face values are similar, so $|\delta_i| \ll |v_i|$.

Quantization Convert floating-point deltas to integers with configurable precision p (decimal digits) using eq. (19):

$$q_i = \text{round}(\delta_i \times 10^p) \quad (19)$$

The precision parameter controls the trade-off between compression ratio and accuracy. With $p = 6$ (default), values are accurate to $\sim 10^{-6}$ relative precision.

Zigzag Encoding Convert signed integers to unsigned for efficient varint encoding (eq. (20)):

$$z_i = \begin{cases} 2q_i & \text{if } q_i \geq 0 \\ -2q_i - 1 & \text{if } q_i < 0 \end{cases} = (q_i \ll 1) \oplus (q_i \gg 31) \quad (20)$$

This maps small-magnitude signed integers to small unsigned integers: $0 \mapsto 0$, $-1 \mapsto 1$, $1 \mapsto 2$, $-2 \mapsto 3$, etc.

Variable-Length Integer Encoding Encode unsigned integers using 7 bits per byte, with the high bit indicating continuation (eq. (21)):

$$\text{bytes}(z) = \left\lceil \frac{\lfloor \log_2(z) \rfloor + 1}{7} \right\rceil \quad (21)$$

Small values ($z < 128$) use 1 byte; larger values use more bytes. Since smooth CFD fields produce small deltas, most values compress to 1–2 bytes instead of 4–8 bytes for raw floats/doubles.

7.2.3 Delta-Varint-Temporal Codec (DVZT)

Enhanced codec that exploits both spatial and temporal correlation for better compression (see figs. 18 to 20):

1. **Keyframes** (every N timesteps): Self-contained, same as DVZ (fig. 18)
2. **Delta frames**: Hybrid spatial-temporal prediction (fig. 19)
 - Uses weighted prediction: $\hat{v} = 0.3 \cdot v_{\text{spatial}} + 0.7 \cdot v_{\text{temporal}}$
 - Encodes residuals (actual - predicted) instead of raw spatial deltas
 - Typically $\sim 10\%$ smaller than DVZ for delta frames

```

1 writeFormat          dvzt;
2 deltaVarintPrecision 6;          // ~1e-6 relative precision
3 dvztKeyframeInterval 20;         // Keyframe every 20 timesteps (default)
```

Listing 25: DVZT configuration

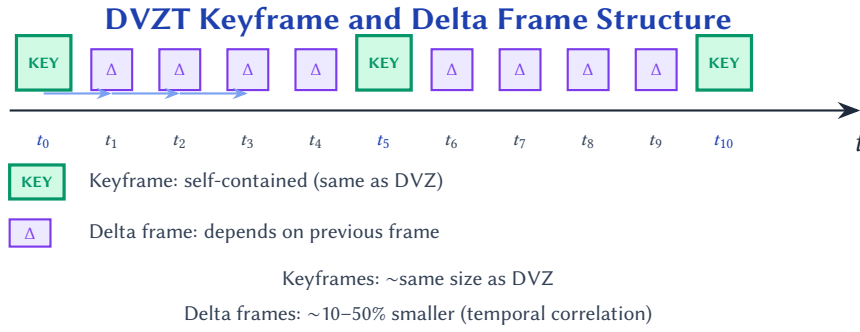


Figure 18: DVZT frame structure: keyframes are self-contained and appear every N timesteps (configurable). Delta frames between keyframes use temporal prediction for better compression

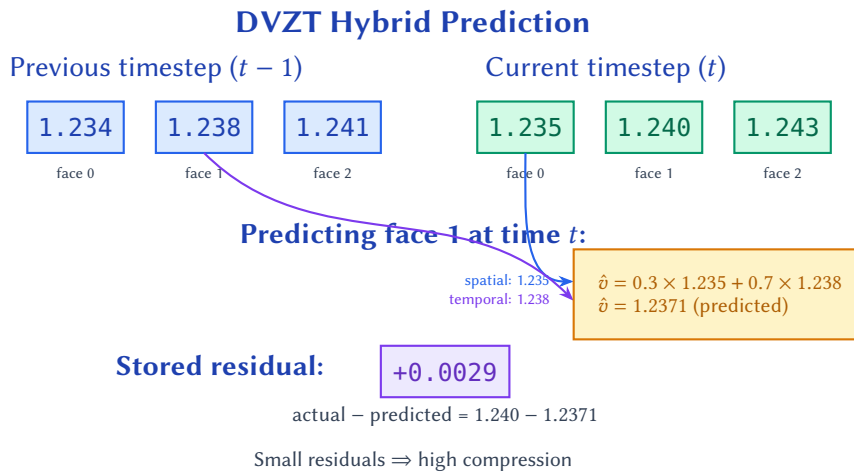


Figure 19: DVZT hybrid prediction: each value is predicted using 30% spatial neighbor (previous face at same timestep) and 70% temporal neighbor (same face at previous timestep). Only the small residual is stored

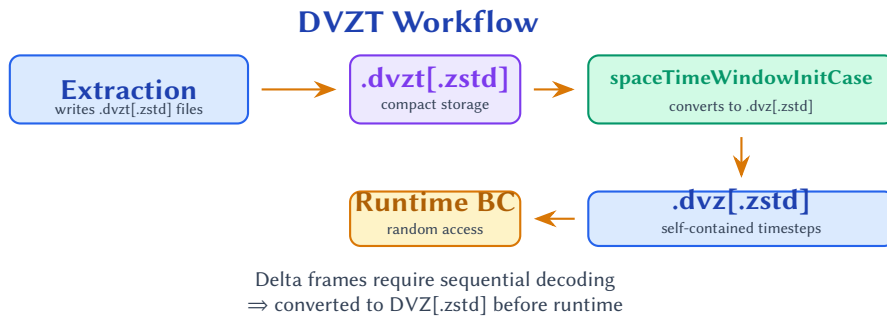


Figure 20: DVZT workflow: extraction writes compact .dvzt[.zstd] files. Before reconstruction, spaceTimeWindowInitCase converts them to .dvz[.zstd] format (required for random timestep access at runtime). The zstd compression layer is preserved through the conversion

7.2.4 DVZT Mathematical Formulation

Frame Types DVZT distinguishes between keyframes and delta frames based on the timestep index k (eq. (22)):

$$\text{Frame type}(k) = \begin{cases} \text{Keyframe} & \text{if } k \bmod K = 0 \\ \text{Delta frame} & \text{otherwise} \end{cases} \quad (22)$$

where K is the keyframe interval (default $K = 20$).

Keyframe Encoding Keyframes use pure DVZ encoding (spatial delta only), as shown in eq. (23):

$$\delta_i^{(k)} = v_i^{(k)} - v_{i-1}^{(k)} \quad (\text{spatial delta}) \quad (23)$$

Delta Frame Hybrid Prediction Delta frames use a weighted combination of spatial and temporal neighbors (eq. (24)):

$$\hat{v}_i^{(k)} = \alpha \cdot v_{i-1}^{(k)} + (1 - \alpha) \cdot v_i^{(k-1)} \quad (24)$$

where $\alpha = 0.3$ (30% spatial weight, 70% temporal weight).

The residual to be encoded is given by eq. (25):

$$r_i^{(k)} = v_i^{(k)} - \hat{v}_i^{(k)} \quad (25)$$

For slowly varying flows with small Δt , consecutive timesteps are nearly identical, so $v_i^{(k)} \approx v_i^{(k-1)}$, making the temporal prediction highly accurate and $|r_i^{(k)}| \ll |v_i^{(k)} - v_{i-1}^{(k)}|$.

Compression Ratio Analysis Let σ_s^2 be the variance of spatial deltas and σ_t^2 the variance of temporal deltas. The hybrid prediction residual variance is approximately (eq. (26)):

$$\sigma_r^2 \approx \alpha^2 \sigma_s^2 + (1 - \alpha)^2 \sigma_t^2 \quad (26)$$

For small timesteps where $\sigma_t^2 \ll \sigma_s^2$ (eq. (27)):

$$\frac{\sigma_r}{\sigma_s} \approx \alpha = 0.3 \quad (27)$$

This explains the $\sim 70\%$ reduction in residual magnitude, which translates to significant compression improvement through varint encoding.

Decoding Dependency Delta frames depend on the previous frame for reconstruction (eq. (28)):

$$v_i^{(k)} = r_i^{(k)} + \alpha \cdot v_{i-1}^{(k)} + (1 - \alpha) \cdot v_i^{(k-1)} \quad (28)$$

This creates a dependency chain from each keyframe (eq. (29)):

$$\text{Keyframe } k_0 \rightarrow \text{Frame } k_0 + 1 \rightarrow \dots \rightarrow \text{Frame } k_0 + K - 1 \quad (29)$$

Random access requires decoding from the nearest preceding keyframe. For this reason, the initialization utility converts DVZT to DVZ before runtime.

DVZT Workflow:

During extraction, DVZT writes smaller `.dvzt` files (or `.dvzt.zstd` with integrated zstd). During case initialization, the utility automatically converts DVZT to DVZ format (required because delta frames need sequential processing). The zstd layer is preserved through the conversion. The resulting `.dvz` (or `.dvz.zstd`) files are read by the spaceTimeWindow BC at runtime.

Extraction → .dvzt[.zstd] files → spaceTimeWindowInitCase → .dvz[.zstd] files → Runtime

When to use DVZT:

- Long simulations with many timesteps (storage savings accumulate)
- Network/disk bandwidth constraints during extraction
- Small timesteps ($\Delta t = 10^{-5}$ or 10^{-6}) where temporal correlation is very strong
- DNS or acoustic simulations requiring fine temporal resolution

Compression vs Timestep Size:

DVZT benefits increase dramatically with smaller timesteps because consecutive values become nearly identical (see table 11):

Table 11: DVZT compression improvement by timestep size

Δt	Temporal Correlation	DVZT vs DVZ Savings
10^{-4}	Moderate	~10% smaller
10^{-5}	Strong	~20–30% smaller
10^{-6}	Very strong	~30–50% smaller

When to use DVZ:

- Simpler workflow (no conversion step)
- When random access to individual timesteps is needed during extraction
- Shorter simulations where DVZT overhead isn't worth it
- Large timesteps where temporal correlation is weak

7.2.5 Integrated Zstd Compression

When built with libzstd support (auto-detected by ./Allwmake), DVZ and DVZT data is automatically wrapped with zstd level-3 compression before writing to disk. This is applied transparently as a compression layer in the encoding pipeline:

codec encode → zstd compress → [optional encrypt] → write to disk

Extension stacking:

- fieldName.dvz.zstd — DVZ with integrated zstd
- fieldName.dvzt.zstd — DVZT with integrated zstd
- fieldName.dvz.zstd.enc — DVZ + zstd + encryption
- fieldName.dvzt.zstd.enc — DVZT + zstd + encryption

The zstd layer provides an additional ~10× reduction on top of the already-compact varint encoding, bringing typical file sizes to ~0.3–0.4% of uncompressed ASCII. Decompression is transparent at runtime—the boundary conditions automatically detect and decompress .zstd files.

i Note

No configuration is needed: zstd compression is applied automatically when the library is built with libzstd. Files without the `.zstd` extension (from older builds or builds without libzstd) remain fully readable for backward compatibility.

The DVZT workflow with integrated zstd becomes:

`.dvzt.zstd` files → `spaceTimeWindowInitCase` → `.dvz.zstd` files → Runtime

7.2.6 External Compression Benchmark**i Note**

With integrated zstd compression (built with libzstd), boundary data is already compressed on the fly with zstd -3. The benchmarks below show what is achievable with *external* tools on *uncompressed* DVZ/DVZT files. For archival of already zstd-compressed files, use 7z LZMA2 for additional gains.

DVZ and DVZT files can be further compressed using external tools for archival or transfer. The following benchmarks compare various compression algorithms on real boundary data files.

Test Environment:

- **CPU:** Intel Core i7-4790 @ 3.60 GHz (4 cores, 8 threads, Haswell microarchitecture)
- **L3 Cache:** 8 MB
- **RAM:** DDR3-1600

Test Data: 618 DVZ files totaling 13 MB (spatial delta-varint encoded), and 938 DVZT files totaling 30 MB (temporal delta-varint encoded).

The compression results are presented in table 12 for DVZ files and table 13 for DVZT files.

Table 12: External compression results for DVZ files (13 MB original)

Method	Size	Ratio	Speed	Notes
7z lzma2 -mx9	769 KB	6.01%	11.2 MB/s	Best ratio
xz -6	769 KB	6.01%	10.2 MB/s	
zstd -ultra -22	964 KB	7.53%	4.7 MB/s	Very slow
zstd -19	965 KB	7.54%	15.3 MB/s	
rar -m5	1017 KB	7.95%	26.9 MB/s	Best balance
7z ppmd -mx9	1.4 MB	11.02%	10.7 MB/s	
zstd -9	1.7 MB	13.17%	109.5 MB/s	
zstd -3	1.8 MB	13.65%	309.0 MB/s	
zstd -1	1.9 MB	14.43%	533.7 MB/s	
bzip2 -9	2.0 MB	15.28%	13.5 MB/s	
gzip -6	2.1 MB	16.39%	94.2 MB/s	
lz4	2.3 MB	17.80%	635.2 MB/s	

Key findings:

Table 13: External compression results for DVZT files (30 MB original)

Method	Size	Ratio	Speed	Notes
7z lzma2 -mx9	1.9 MB	6.31%	10.1 MB/s	Best ratio
xz -6	1.9 MB	6.32%	10.5 MB/s	
zstd -ultra -22	2.6 MB	8.48%	2.4 MB/s	Very slow
zstd -19	2.6 MB	8.50%	10.4 MB/s	
rar -m5	2.8 MB	9.28%	28.0 MB/s	
zstd -9	2.8 MB	9.27%	120.6 MB/s	
7z ppmD -mx9	2.8 MB	9.37%	12.2 MB/s	
zstd -3	3.1 MB	10.10%	396.1 MB/s	Best balance
zstd -1	3.1 MB	10.25%	614.0 MB/s	
bzip2 -9	3.5 MB	11.52%	13.7 MB/s	
gzip -6	3.8 MB	12.71%	104.3 MB/s	
lz4	4.1 MB	13.56%	763.0 MB/s	Fastest

- **Best compression ratio:** 7z LZMA2 and xz achieve ~6% (94% reduction)
- **Best speed/ratio balance:** zstd -3 at 10–14% ratio with 300–400 MB/s throughput
- **zstd -3 is 40× faster than xz** with only 60% more space
- **zstd -ultra -22 provides no benefit** over zstd -19 for this data type
- **bzip2 and PPMd perform poorly** for CFD boundary data

Recommendations:

- **Runtime/on-the-fly:** Use integrated zstd (build with libzstd, automatic)
- **Archive/transfer:** 7z lzma2 -mx9 or xz -6 (~6% ratio, 10 MB/s)
- **Real-time streaming:** lz4 (~14–18% ratio, 600–800 MB/s)

```

1 # Archive with best compression (7z)
2 cd subset-case/constant/boundaryData/oldInternalFaces
3 7z a -m0=lzma2 -mx=9 ../boundaryData.7z */*.dvz.zstd */*.dvz */*.dvzt.zstd
   */*.dvzt
4
5 # Or with tar (files already zstd-compressed if built with libzstd)
6 tar -cf ../boundaryData.tar */*.dvz.zstd

```

Listing 26: Archival compression example

7.3 Extraction Metadata

The extractionMetadata file contains:

```

1 {
2   openfoamVersion    "v2512";
3   openfoamApi        2512;

```

```

4     solver          "pimpleFoam";
5     deltaT          1e-04;
6     adjustTimeStep  false;
7     timePrecision   6;
8     extractionStartTime 0.0001;
9     boxMin          (0.05 -0.25 0.01);
10    boxMax          (0.90 0.25 0.38);
11    nGlobalFaces     12345;
12    extractGhostCells true;          // present when ghost cells enabled
13    nGhostCells      432;          // number of ghost cells
14    timesteps        (0.0001 0.0002 0.0003 ...);
15 }

```

Listing 27: extractionMetadata contents

8 Parallel Execution

Both the source simulation (extraction phase) and the reconstruction simulation can run in parallel. This enables efficient use of HPC resources for both phases of the workflow.

8.1 Parallel Extraction

The extraction function object fully supports parallel execution:

- Extraction box can span multiple processor domains
- Boundary data is gathered from all processors
- Master processor writes combined data files
- Processor boundary faces are handled automatically

```

1  # Decompose the case
2  decomposePar
3
4  # Run parallel extraction
5  mpirun -np 8 pimpleFoam -parallel
6
7  # Automatic field writes occur at:
8  #   t_0 - extraction start (for lookback)
9  #   t_1 - linear interpolation start
10 #   t_2 - cubic interpolation start

```

Listing 28: Parallel extraction

8.2 Field Reconstruction

After parallel extraction, reconstruct fields before case initialization:

```

1  # For cubic interpolation (recommended)
2  reconstructPar -time <t_2>

```

```

3
4 # For linear interpolation
5 reconstructPar -time <t_1>

```

Listing 29: Field reconstruction

△ Warning

The subset mesh created by `spaceTimeWindowInitCase` uses cell ordering from the reconstructed (serial) source mesh. Running `reconstructPar` before case initialization is mandatory for parallel extractions.

8.3 Parallel Reconstruction

The reconstruction simulation can also run in parallel, independently of how the extraction was performed:

```

1 cd subset-case
2 decomposePar
3 mpirun -np 4 pimpleFoam -parallel
4 reconstructPar

```

Listing 30: Parallel reconstruction

✓ Tip

The reconstruction case is typically much smaller than the source case, so fewer processors may be needed. The `spaceTimeWindow` boundary conditions work identically in serial and parallel modes.

9 Mass Conservation

9.1 The Mass Imbalance Problem

Face interpolation during extraction can introduce small mass flux imbalances:

$$\text{imbalance} = \sum_f \mathbf{U}_f \cdot \mathbf{S}_f \neq 0 \quad (30)$$

When all boundary patches have `fixesValue=true`, `adjustPhi()` cannot correct this imbalance, potentially causing pressure solver issues. Additionally, with all-Dirichlet BCs, the solver has no way to relieve pressure buildup.

9.2 Solutions

9.2.1 Use Pressure-Coupled BC (Recommended)

The `spaceTimeWindowCoupledPressure` BC handles mass conservation while accurately reproducing the original pressure field:

- Uses mixed Dirichlet/Neumann based on local flux

- Two modes: `fluxMagnitude` (default, Dirichlet at stagnant faces) or `flowDirection` (Dirichlet at inflow)
- Neumann faces allow natural pressure adjustment for continuity compatibility
- Requires gradient extraction during source simulation

See section 4.1 for detailed configuration.

9.2.2 Use `-inletOutletBC` (for quick tests)

The flux-based inlet-outlet BC naturally handles mass conservation:

- Outflow faces use `zeroGradient`, allowing natural outflow
- No artificial mass imbalance from prescribed outflow velocities
- Works without any special mass correction

```
1 spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC
```

Listing 31: Using inlet-outlet BC for mass conservation

9.2.3 Use `-correctMassFlux`

Applies least-squares correction to boundaryData to ensure exact mass conservation:

$$\mathbf{U}_{\text{corrected}} = \mathbf{U} - \frac{\text{imbalance}}{\sum_f |\mathbf{S}_f|} \cdot \hat{\mathbf{n}} \quad (31)$$

where $\hat{\mathbf{n}} = \mathbf{S}_f / |\mathbf{S}_f|$ is the face unit normal.

This minimizes $\|\mathbf{U}_{\text{corrected}} - \mathbf{U}\|^2$ subject to:

$$\sum_f \mathbf{U}_{\text{corrected}} \cdot \mathbf{S}_f = 0 \quad (32)$$

```
1 spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -correctMassFlux
```

Listing 32: Using mass flux correction

9.2.4 Use `-outletDirection with fixesValue true`

Creates an outlet patch where mass imbalance can escape:

- `oldInternalFaces` uses `fixesValue true` (values not modified by `adjustPhi`)
- `outlet` uses `inletOutlet` BC (allows `adjustPhi` correction)

9.3 The fixesValue Option

The `fixesValue` parameter controls `adjustPhi()` behavior:

- `fixesValue = true`: Patch excluded from flux correction (preserves exact values)
- `fixesValue = false`: Patch included in flux correction (allows modification)

9.4 Outlet Patch for Pressure Relief

The `-outletDirection` option creates an outlet patch (see fig. 21):

```
1 spaceTimeWindowInitCase -sourceCase ../source \
2   -outletDirection "(1 0 0)" \
3   -outletFraction 0.1
```

Listing 33: Creating outlet patch

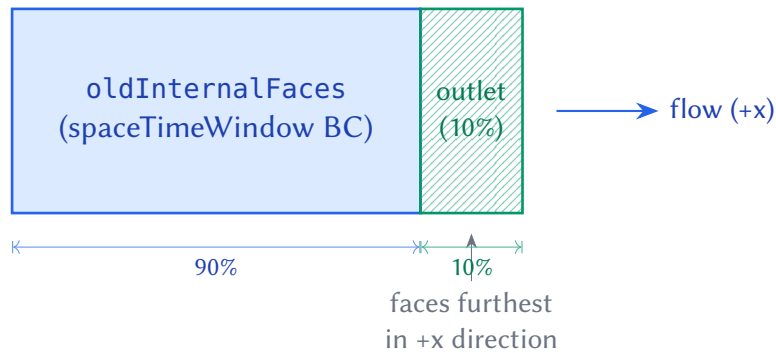


Figure 21: Outlet patch creation with `-outletDirection "(1 0 0)"`

Outlet faces are selected based on position (furthest along the outlet direction), not face orientation.

9.5 Recommended Configurations

The recommended mass conservation strategies are summarized in table 14.

Table 14: Mass conservation strategies

Configuration	Description
<code>-ghostCells</code>	Machine-precision. Eliminates boundary stencil mismatch by including ghost cells. Requires <code>pimpleFoamStag</code> or <code>pimpleFoamPrecise</code> .
<code>-preciseBC</code>	Highest fidelity. Most accurate pressure recovery using <code>HbyA (u*)</code> . Requires <code>pimpleFoamPrecise</code> extraction.
<code>-ghostCells + -preciseBC</code>	Maximum fidelity. Ghost cells + u^* boundary data.
<code>-inletOutletBC</code>	Recommended for quick tests. Natural mass balance through flux-based switching.
<code>-inletOutletBC -correctMassFlux</code>	Unsteady turbulent + extra safety. Ensures exact mass balance with inlet-outlet switching.
<code>-outletDirection "(1 0 0)"</code>	Steady-mean flow direction. Use when outlet location is known and flow direction is consistent.
<code>-correctMassFlux (no outlet)</code>	All faces prescribed with exact mass balance.

10 Solver Settings and Relaxation

10.1 Time Stepping

For accurate reconstruction, use identical time stepping:

```

1  deltaT          1e-04;      // Must match extraction
2  adjustTimeStep  no;         // Fixed timestep recommended
3
4  // If adaptive timestep is required:
5  adjustTimeStep  yes;
6  maxCo           0.5;
7  maxDeltaT       1e-03;
```

Listing 34: Recommended controlDict settings

⚠ Warning

The reconstruction must use identical `deltaT` and `adjustTimeStep` settings as the extraction. Mismatches cause fatal errors unless `allowTimeInterpolation=true`.

10.2 PIMPLE Settings

For incompressible flows:

```

1  PIMPLE
2  {
3      nOuterCorrectors    2;
4      nCorrectors         2;
5      nNonOrthogonalCorrectors 1;
6
7      // Pressure reference (set by spaceTimeWindowInitCase)
8      pRefPoint            (0.5 0 0.2);
9      pRefValue            0;
10 }
```

Listing 35: Recommended PIMPLE settings

10.3 Relaxation Factors

✓ Tip

For reconstruction simulations, relaxation is typically **not needed** because:

- Boundary conditions are prescribed (not coupled)
- Initial conditions come from the source simulation
- Flow field evolves smoothly from physical initial state

If stability issues occur, try:

```

1 relaxationFactors
2 {
3     fields
4     {
5         p                0.7;
6     }
7     equations
8     {
9         U                0.7;
10        "(k|epsilon|omega|nuTilda)" 0.7;
11    }
12 }

```

Listing 36: Optional relaxation

10.4 Linear Solver Settings

Standard settings work well:

```

1 solvers
2 {
3     p
4     {
5         solver            GAMG;
6         smoother          GaussSeidel;
7         tolerance         1e-06;
8         relTol            0.01;
9     }
10
11    U
12    {
13        solver            PBiCGStab;
14        preconditioner     DILU;
15        tolerance         1e-06;
16        relTol            0.1;
17    }
18 }

```

Listing 37: Linear solver settings

11 Time Interpolation

The boundary conditions support three time interpolation modes, summarized in Table 15 and illustrated in fig. 22. The comparison between linear and cubic interpolation quality is shown in fig. 23.

`spaceTimeWindowInitCase` automatically sets `startTime` and `endTime` to ensure sufficient buffer timesteps for the selected interpolation scheme (see fig. 24 for buffer requirements).

11.1 Exact Timestep Matching (Default)

By default, the boundary condition requires exact timestep matching:

Table 15: Time interpolation modes

Mode	Start Time	Timesteps Used	Use Case
none (exact)	t_0	1 (exact match)	Highest fidelity (no interpolation error)
linear	t_1	2 (bracketing)	Simple, smoothly varying flows
cubic	t_2	4 (Catmull-Rom)	Unsteady turbulent flows (recommended)

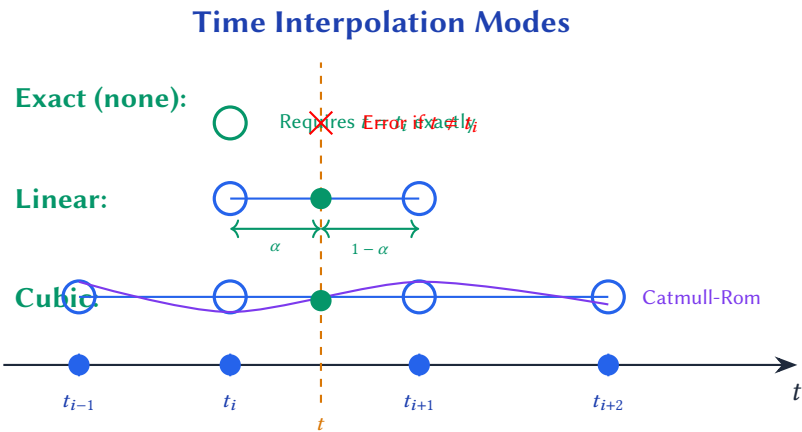


Figure 22: Time interpolation modes: exact matching uses only one timestep, linear interpolation uses two bracketing timesteps, and cubic (Catmull-Rom) uses four timesteps for smooth C^1 continuous interpolation

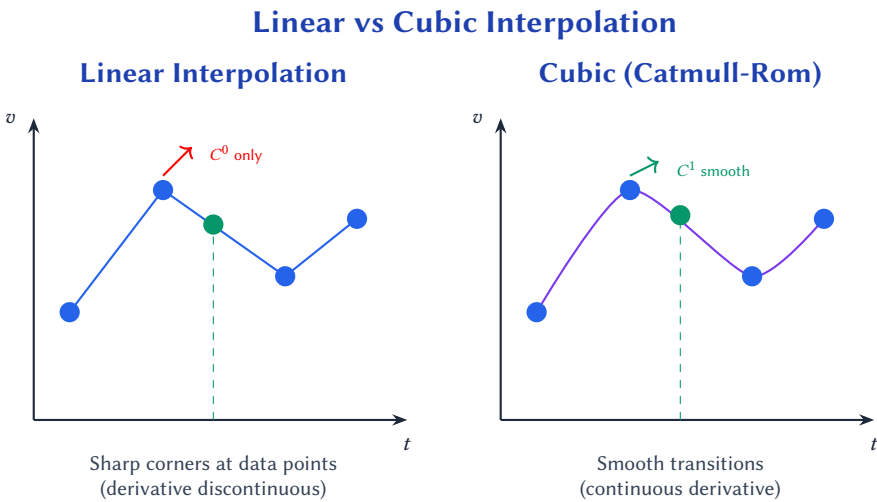


Figure 23: Comparison of linear and cubic interpolation: linear interpolation creates C^0 continuous curves with sharp corners at data points, while Catmull-Rom cubic splines provide C^1 continuity with smooth transitions

Time Buffer Requirements

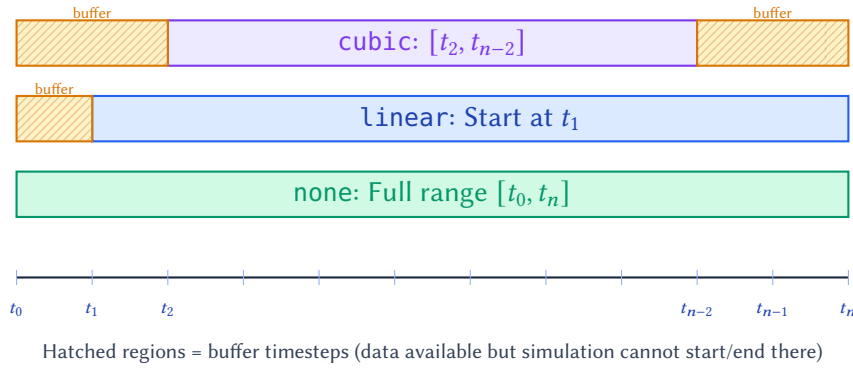


Figure 24: Time buffer requirements for each interpolation mode: exact matching can use the full range, linear needs one buffer timestep at the start, and cubic needs two buffer timesteps at both start and end

- Simulation time must match available sample times (within 1% of Δt)
- Fatal error if no matching timestep found
- Ensures highest fidelity (no time interpolation error)

11.2 Linear Interpolation

For cases where exact matching is not possible:

```

1 oldInternalFaces
2 {
3     type                spaceTimeWindow;
4     allowTimeInterpolation true;
5     timeInterpolationScheme linear;
6     // ...
7 }
```

Listing 38: Linear interpolation

Uses two bracketing timesteps (eq. (33)):

$$\text{value} = (1 - \alpha) \cdot v_i + \alpha \cdot v_{i+1} \quad \text{where} \quad \alpha = \frac{t - t_i}{t_{i+1} - t_i} \quad (33)$$

The interpolation parameter $\alpha \in [0, 1]$ represents the normalized position between the two samples. This is equivalent to first-order polynomial interpolation (eq. (34)):

$$\phi(t) = \phi_i + \frac{\phi_{i+1} - \phi_i}{t_{i+1} - t_i} (t - t_i) \quad (34)$$

Linear interpolation provides C^0 continuity (continuous values but discontinuous derivatives at sample points).

11.3 Cubic Spline Interpolation

For smoother results with adaptive timestepping:

```

1 oldInternalFaces
2 {
3     type                spaceTimeWindow;
4     allowTimeInterpolation true;
5     timeInterpolationScheme cubic;
6     // ...
7 }
```

Listing 39: Cubic interpolation

Uses centripetal Catmull-Rom spline with four timesteps ($t_{i-1}, t_i, t_{i+1}, t_{i+2}$):

- Handles non-uniform time spacing correctly
- Provides C^1 continuity
- No overshoots or cusps
- Essential for `adjustTimeStep=yes`

11.3.1 Catmull-Rom Spline Formulation

The centripetal Catmull-Rom spline passes through all control points and provides C^1 continuity. Given four consecutive sample values $\phi_0, \phi_1, \phi_2, \phi_3$ at times t_0, t_1, t_2, t_3 , the interpolated value for $t \in [t_1, t_2]$ is computed as follows.

First, compute the centripetal parameterization distances (eq. (35)):

$$d_j = |t_{j+1} - t_j|^{0.5}, \quad j = 0, 1, 2 \quad (35)$$

The normalized parameter within the segment $[t_1, t_2]$ is (eq. (36)):

$$u = \frac{t - t_1}{t_2 - t_1} \quad (36)$$

The spline is constructed using the Barry-Goldman algorithm. Define intermediate points (eq. (37)):

$$\begin{aligned} A_1 &= \frac{t_2 - t}{t_2 - t_0} \phi_0 + \frac{t - t_0}{t_2 - t_0} \phi_1 \\ A_2 &= \frac{t_2 - t}{t_2 - t_1} \phi_1 + \frac{t - t_1}{t_2 - t_1} \phi_2 \\ A_3 &= \frac{t_3 - t}{t_3 - t_1} \phi_2 + \frac{t - t_1}{t_3 - t_1} \phi_3 \end{aligned} \quad (37)$$

Then compute (eq. (38)):

$$\begin{aligned} B_1 &= \frac{t_2 - t}{t_2 - t_0} A_1 + \frac{t - t_0}{t_2 - t_0} A_2 \\ B_2 &= \frac{t_3 - t}{t_3 - t_1} A_2 + \frac{t - t_1}{t_3 - t_1} A_3 \end{aligned} \quad (38)$$

Finally, the interpolated value is (eq. (39)):

$$\phi(t) = \frac{t_2 - t}{t_2 - t_1} B_1 + \frac{t - t_1}{t_2 - t_1} B_2 \quad (39)$$

This formulation correctly handles non-uniform time spacing, which is essential when adaptive timestepping (`adjustTimeStep=yes`) produces variable timesteps.

11.4 Time Range Requirements

- **Linear:** Needs 2 buffer timesteps at start
- **Cubic:** Needs 2 buffer timesteps at start and end
- `spaceTimeWindowInitCase` automatically sets appropriate `startTime` and `endTime`

i Note

Extrapolation outside the extraction window is **never** allowed, regardless of interpolation settings.

12 Flux Reporting and Diagnostics

Enable flux reporting to monitor mass conservation:

```
1 oldInternalFaces
2 {
3     type          spaceTimeWindow;
4     reportFlux    true;
5     // ...
6 }
```

Listing 40: Enabling flux reporting

Output format (fields explained in table 16):

```
1 spaceTimeWindow flux [oldInternalFaces] t=0.001 thisPatch=1.234e-06 (in=-0.0523 out=0.0523) | MESH TOTAL=5.678e-05 (fixed=-0.0523
adjustable=0.0523) | adjustPhi will correct: -5.678e-05
```

Listing 41: Flux report output

Table 16: Flux report fields

Field	Description
<code>thisPatch</code>	Net flux through this <code>spaceTimeWindow</code> patch
<code>in</code>	Sum of inward fluxes
<code>out</code>	Sum of outward fluxes
<code>MESH TOTAL</code>	Total net flux across all boundary patches
<code>fixed</code>	Flux from patches with <code>fixesValue=true</code>
<code>adjustable</code>	Flux from patches with <code>fixesValue=false</code>
<code>adjustPhi will correct</code>	Correction to be distributed

13 Troubleshooting

13.1 Common Errors

13.1.1 Time Step Mismatch

```

1 FOAM FATAL ERROR:
2 Time step mismatch between extraction and reconstruction!
3   Extraction deltaT: 8.53e-04
4   Current deltaT:    1e-03

```

Listing 42: deltaT mismatch error

Solution: Set deltaT in system/controlDict to match extraction.

13.1.2 No Matching Timestep

```

1 FOAM FATAL ERROR:
2 No exact timestep match found for t = 0.00015
3 Available times: 0.0001, 0.0002, 0.0003

```

Listing 43: Missing timestep error

Solution: Enable time interpolation with allowTimeInterpolation true.

13.1.3 Pressure Solver Divergence

Symptoms: Pressure residuals increase, NaN values appear.

Solutions:

1. Use -correctMassFlux option
2. Add outlet patch with -outletDirection
3. Set fixesValue false on oldInternalFaces
4. Check pRefPoint is inside the domain

13.1.4 Encryption Errors

```

1 FOAM FATAL ERROR:
2 Failed to decrypt file: constant/boundaryData/.../U.dvz.zstd.enc

```

Listing 44: Decryption failure

Solutions:

1. Verify private key is correct
2. Ensure library was built with FOAM_USE_SODIUM=1
3. Check file is not corrupted

14 Acknowledging this work

If this software has been useful in your research, please consider citing:

Anton, A.-A. (2011). “Space-Time Window Reconstruction in Parallel High Performance Numeric Simulations. Application for CFD”, PhD Thesis, Politehnica University of Timisoara. Available at: <https://dspace.upt.ro/jspui/handle/123456789/643>

```

1 @phdthesis{Anton2011,
2   author   = "Alin-Adrian Anton",
3   title    = "Space-Time Window Reconstruction in High-Performance
4               Numeric Simulations: Application for {CFD}",
5   school   = "Universitatea Politehnica Timi\c{s}oara",
6   year     = "2011",
7   month    = "November",
8   type     = "PhD thesis",
9   address  = "Timi\c{s}oara, Romania",
10  publisher = "Editura Politehnica",
11  isbn      = "978-606-554-390-4",
12  url       = "https://dspace.upt.ro/jspui/handle/123456789/643"
13 }

```

Listing 45: BibTeX entry

Or acknowledge: “This work made use of the openfoam-spaceTimeWindow library available at <https://github.com/cherubimro/openfoam-spaceTimeWindow/>.”

Secondary link: <https://dev.cs.upt.ro/alin.anton/openfoam-spaceTimeWindow>.

15 Acknowledgments

This library implements the space-time window reconstruction concept [2, 3].

The example case uses mesh generation from the ERCOFTAC Classic Collection Database [9].

The square cylinder benchmark setup was published by Niklas Nordin [10].

A Example Case: ERCOFTAC UFR2-02 Square Cylinder

The library includes a complete example based on the ERCOFTAC UFR2-02 benchmark case [9, 11] (Case 043 in the ERCOFTAC database): turbulent flow around a square cylinder at $Re = 21,400$. This case demonstrates vortex shedding and is ideal for testing space-time window reconstruction.

OPENFOAM® is a registered trademark of OpenCFD Limited.

A.1 Reference Data and Validation Resources

The ERCOFTAC (European Research Community on Flow, Turbulence and Combustion) Classic Collection Database provides extensive reference data for this test case, including:

- Experimental measurements from Lyn et al. [11] (laser-Doppler velocimetry)
- Reference numerical simulations from various research groups
- Mesh generation scripts and case setup files
- Detailed flow statistics and validation data

The database is accessible at:

<http://cfd.mace.manchester.ac.uk/ercoftac/>

An OpenFOAM-specific implementation guide is available at:

https://openfoamwiki.net/index.php?title=Benchmark_ercoftac_ufr2-02

i Note

If HTTPS access fails for the ERCOFTAC database, use HTTP (<http://>) instead. The ERCOFTAC server may not support secure connections.

A.2 Physical Background: Von Kármán Vortex Street

When fluid flows past a bluff body (such as a square cylinder), the flow separates at the sharp edges, creating alternating vortices that are shed from each side of the body. This phenomenon, known as the **von Kármán vortex street**, is characterized by:

- **Periodic vortex shedding:** Vortices detach alternately from upper and lower surfaces
- **Strouhal number:** $St = fD/U_\infty \approx 0.13$ for square cylinders, where f is the shedding frequency, D is the cylinder side length, and U_∞ is the freestream velocity
- **Turbulent wake:** At $Re = 21,400$, the wake is fully turbulent with complex three-dimensional structures
- **Unsteady forces:** Fluctuating lift and drag on the cylinder

The vortex street extends far downstream, making it an excellent test case for space-time window extraction—the extraction region can capture the wake dynamics while excluding the cylinder and inlet regions.

A.3 Domain and Mesh Configuration

The computational domain (fig. 25) extends from $-4D$ upstream to $+15D$ downstream of the cylinder, with lateral boundaries at $\pm 6.5D$. The spanwise extent is $4D$ with periodic boundary conditions.

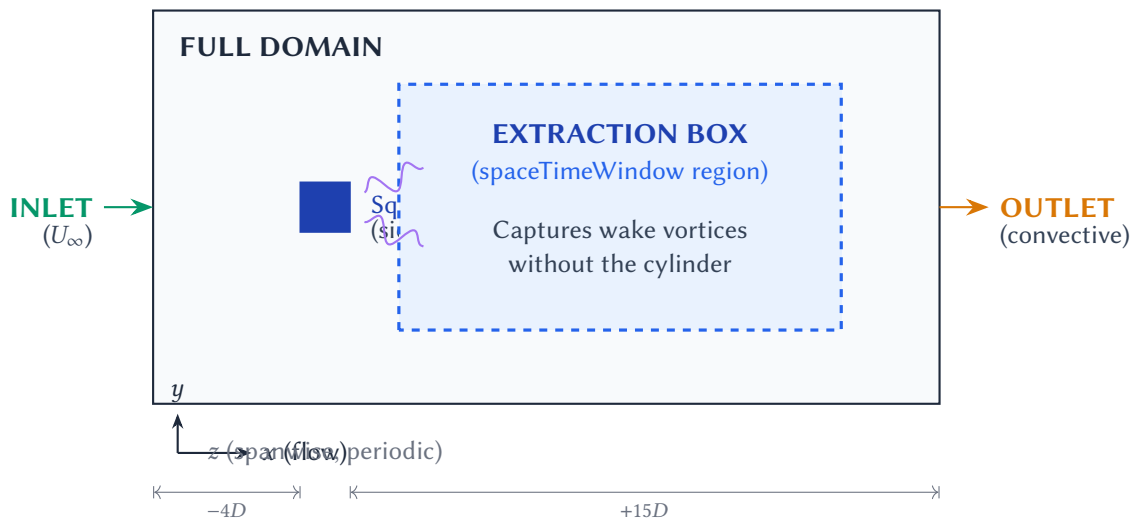


Figure 25: Computational domain for the square cylinder case showing the extraction box placement in the wake region

A.4 Extraction Box Placement

The extraction box is positioned to capture the vortex street while avoiding:

- The cylinder itself (to avoid complex geometry in subset)
- The inlet region (where flow is uniform and uninteresting)
- Domain boundaries (box must be fully internal)

Example extraction configuration for this case:

```

1 extractSubset
2 {
3     type          spaceTimeWindowExtract;
4     libs          (spaceTimeWindow);
5
6     // Box starts 0.5D downstream of cylinder, extends to 9D
7     // Lateral extent captures the wake width
8     // Full spanwise extent (periodic direction)
9     box           ((0.05 -0.25 0.01) (0.90 0.25 0.38));
10
11     outputDir     "../cylinder-subset";
12     fields        (U p nut);
13     writeFormat    deltaVarint;
14
15     writeControl   timeStep;
16     writeInterval  1;
17 }

```

Listing 46: Extraction configuration for square cylinder

A.5 Physical Parameters

The physical parameters for this case are listed in table 17.

Table 17: Square cylinder case parameters

Parameter	Value	Description
D	0.04 m	Cylinder side length
U_∞	8.25 m/s	Freestream velocity
ν	$1.5 \times 10^{-5} \text{ m}^2/\text{s}$	Kinematic viscosity
Re	21,400	Reynolds number
St	≈ 0.13	Strouhal number
T_{shed}	$\approx 0.037 \text{ s}$	Vortex shedding period

A.6 Running the Example

```

1 cd examples/ufr2-02
2

```

```

3 # Generate mesh and run full simulation
4 ./Allrun
5
6 # The extraction creates ../ufr2-02-subset with:
7 #   - Subset mesh in constant/polyMesh
8 #   - Boundary data in constant/boundaryData
9 #   - Initial fields
10
11 # Initialize and run reconstruction (recommended: inlet-outlet BC)
12 cd ../ufr2-02-subset
13 spaceTimeWindowInitCase -sourceCase ../ufr2-02 -inletOutletBC
14
15 pimpleFoam
16
17 # Alternative: fixed outlet direction for steady-mean flows
18 # spaceTimeWindowInitCase -sourceCase ../ufr2-02 \
19 #   -outletDirection "(1 0 0)" \
20 #   -correctMassFlux

```

Listing 47: Running the example case

A.7 Validation

A.7.1 CFD Validation Philosophy

In computational fluid dynamics, validation means demonstrating that a simulation reproduces the *physics* of the flow—not that it produces bit-identical numbers. A validated CFD result agrees with experimental measurements (velocity profiles, pressure distributions, turbulent statistics, shedding frequencies, etc.) within quantified uncertainties. Different meshes, turbulence models, numerical schemes, and even different software packages will produce numerically different fields, yet all of them can be equally “valid” if they capture the physical phenomena to the required accuracy.

This distinction is central to the space-time window approach. The reconstruction does not aim to be a bit-exact copy of the original simulation; it aims to be a *physically faithful* re-simulation of the same flow inside the region of interest. The governing equations, boundary conditions, and fluid properties are identical; what changes is the domain extent and, optionally, the mesh resolution. The reconstructed fields are therefore validated in the same way as any other CFD result: by comparing physical quantities against reference data (experiment or a trusted full-domain simulation).

A concrete example of this validation philosophy in OpenFOAM is the Timișoara Swirl Generator [12], a turbomachinery test rig developed at Politehnica University of Timișoara featuring guide vanes, a free runner with 10 blades, and a convergent-divergent draft tube. Petit et al. performed unsteady simulations using OpenFOAM’s transientSimpleDyMFoam solver on a block-structured hexahedral mesh with GGI interfaces, and validated the results against Laser Doppler Velocimetry (LDV) measurements and unsteady pressure recordings at multiple stations in the draft tube [12]. The validation criteria were physical: mean and fluctuating velocity profiles, pressure recovery, and vortex rope frequency—not numerical identity with any previous computation. This case, documented on the OpenFOAM Wiki¹, illustrates the general principle that CFD validation is always performed against the physics of the flow.

¹https://openfoamwiki.net/index.php/Sig_Turbomachinery/_Timisoara_Swirl_Generator

A.7.2 Validating the Reconstruction

The reconstruction can be validated by comparing:

- Instantaneous velocity fields at matching timesteps
- Vortex shedding frequency (should match original)
- Mean velocity profiles in the wake
- Reynolds stress components

Because the reconstruction solves the same equations with the same physical parameters, any discrepancy with the full-domain solution is due to the finite-volume discretisation change at the extraction boundary (see section 2.6), not to different physics. These errors are localised near the boundary and decay rapidly with distance, as demonstrated by the accuracy comparison in section 4.3.

i Note

The reconstruction should closely reproduce the original flow within the extraction region (to solver tolerance) when using exact timestep matching. Small differences may occur at the outlet boundary due to the different boundary condition type. When the mesh resolution is changed (refinement or coarsening), additional spatial interpolation errors appear, but the physical behaviour—vortex dynamics, mean profiles, spectral content—is preserved.

B Field Selection by Turbulence Model

When configuring the extraction, include all fields required by your turbulence model. Table 18 lists recommended fields for common models.

Table 18: Recommended fields by turbulence model

Turbulence Model	Recommended Fields
LES Smagorinsky	(U p nut)
LES dynamicKEqn	(U p nut k)
LES WALE	(U p nut)
RANS k- ϵ	(U p nut k epsilon)
RANS k- ω SST	(U p nut k omega)
Spalart-Allmaras	(U p nut nuTilda)

C The pimpleFoamPrecise Solver

The pimpleFoamPrecise solver is a modified version of OpenFOAM's standard pimpleFoam that registers the intermediate velocity $\mathbf{u}^* = H(\mathbf{U})/A$ (named HbyA) in the objectRegistry, making it accessible to function objects such as spaceTimeWindowExtract.

C.1 Motivation

In the PIMPLE algorithm, the momentum equation is semi-discretized as:

$$AU = H(\mathbf{U}) - \nabla p$$

The intermediate velocity $\mathbf{u}^* = H(\mathbf{U})/A$ is computed before the pressure correction step. The pressure Poisson equation is then:

$$\nabla \cdot \left(\frac{1}{A} \nabla p \right) = \nabla \cdot \mathbf{u}^*$$

Wu, Zaki, and Meneveau [8] in 2020 showed that using \mathbf{u}^* instead of the final divergence-free velocity \mathbf{U} as the boundary condition for subdomain reconstruction yields the highest fidelity pressure recovery. In their DNS context (very fine grids), the high resolution minimizes the non-linear discretization stencil mismatch between full-domain internal faces and subdomain boundary faces [5, 6], and the use of spectral/direct (non-iterative) pressure solvers yields exact solutions, together achieving machine-precision accuracy. On coarser LES/RANS meshes with iterative solvers (e.g. GAMG), the mismatch is larger, though it remains negligible compared to the inherent modelling error of turbulence closures. \mathbf{u}^* remains the theoretically optimal choice: it is the source term of the pressure equation, so prescribing it on the boundary makes the subdomain pressure equation consistent with the full-domain one. The boundary data uses OpenFOAM's exact distance-weighted interpolation formula, and when extraction is performed at every solver timestep no temporal interpolation is needed.

C.2 Implementation

The solver differs from standard `pimpleFoam` in two files:

- **createFields.H:** Declares a persistent `volVectorField` named `HbyA_stored` with `IOobject::REGISTER` that survives across the time loop. This is necessary because the local `HbyA` variable in `pEqn.H` is stack-allocated and its destructor unregisters it from the `objectRegistry` before function objects execute.
- **pEqn.H:** Copies the computed `HbyA` to the persistent field via `HbyA_stored = HbyA` at every PIMPLE pressure correction iteration.

```

1 // Persistent HbyA field for function object access
2 volVectorField HbyA_stored
3 (
4     IOobject
5     (
6         "HbyA",
7         runTime.timeName(),
8         mesh,
9         IOobject::NO_READ,
10        IOobject::NO_WRITE,
11        IOobject::REGISTER
12    ),
13    mesh,
14    dimensionedVector(dimVelocity, Zero)
15 );

```

Listing 48: Persistent `HbyA` field in `createFields.H`

△ Warning

The `IObject::REGISTER` flag is essential in OpenFOAM v2512. Without it, the field is not found by `mesh.findObject<volVectorField>("HbyA")` in the extractor function object.

C.3 Usage

Use `pimpleFoamPrecise` as a drop-in replacement for `pimpleFoam` during the extraction phase:

```
1 # In source case controlDict: application pimpleFoamPrecise;
2 # Extraction function object uses: fields (HbyA p);
3 cd source-case
4 pimpleFoamPrecise
```

Listing 49: Using `pimpleFoamPrecise` for extraction

The reconstruction phase uses standard `pimpleFoam` — no special solver is needed for reconstruction.

i Note

`pimpleFoamPrecise` adds negligible overhead compared to standard `pimpleFoam`. The only additional cost is a single field copy (`HbyA_stored = HbyA`) per pressure correction iteration.

D The pimpleFoamStag Solver

The `pimpleFoamStag` solver implements a Mahesh-style [13] hybrid staggered mesh approach for incompressible, turbulent flow. Face-normal velocity U_n is the primary kinematic variable, eliminating Rhie-Chow interpolation [14] and providing exact discrete mass conservation.

D.1 Algorithm

The algorithm per PIMPLE iteration is:

1. **Reconstruct:** Cell-centred U from face-normal U_n via least-squares
2. **Momentum:** Cell-centred $UEqn$ using standard `fvm/fvc` operators
3. **Project:** $HbyA$ to face-normal U_n^*
4. **Pressure:** $\nabla \cdot (r_{AU_f} \nabla p) = \nabla \cdot \phi^*$
5. **Correct:** $U_n = U_n^* - r_{AU_f} \partial p / \partial n$ (exact, no Rhie-Chow)
6. **Turbulence:** `turbulence->correct()` on final iteration

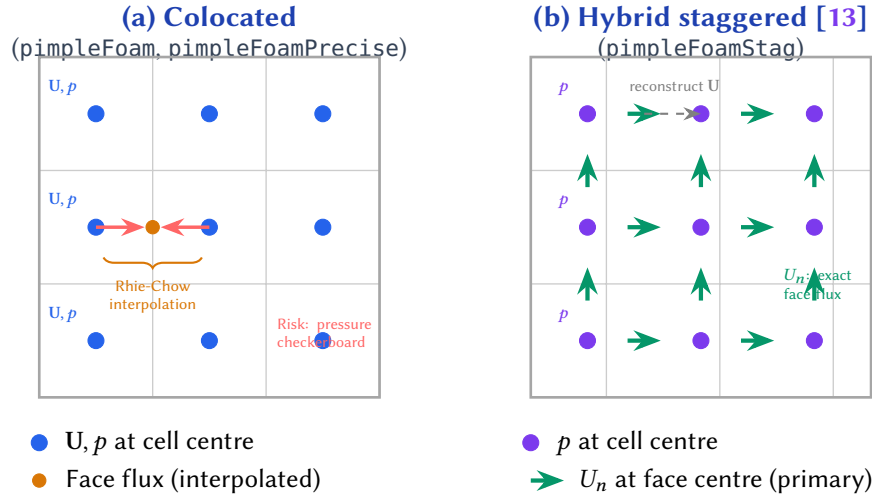


Figure 26: Comparison of variable arrangements in the finite volume method. (a) Colocated: all variables (U, p) stored at cell centres; face velocities for flux computation require Rhie-Chow interpolation to suppress pressure–velocity decoupling (checkerboard oscillations). (b) Hybrid staggered [13]: face-normal velocity U_n is the primary kinematic variable stored directly at face centres; pressure p at cell centres. Face flux is computed exactly from U_n without interpolation, inherently preventing checkerboard modes. Cell-centred U is reconstructed from U_n via least-squares each iteration.

D.2 Colocated vs Staggered Discretization

The fundamental difference between `pimpleFoamPrecise` (and standard `pimpleFoam`) and `pimpleFoamStag` lies in where the velocity unknowns are stored, as illustrated in fig. 26.

Colocated (standard FVM): In the colocated arrangement used by `pimpleFoam` and `pimpleFoamPrecise`, all field variables (U, p) are stored at cell centres. When computing the face flux for the continuity equation, cell-centred velocities must be interpolated to face centres. Naive linear interpolation leads to *pressure–velocity decoupling*: the pressure field can develop a checkerboard pattern (alternating high/low values) that satisfies the discrete equations but is physically meaningless. The Rhie-Chow interpolation [14] adds a pressure-dependent correction to the face velocity:

$$U_f = \bar{U}_f - \left(\frac{1}{A} \right)_f \left[(\nabla p)_f - \overline{(\nabla p)}_f \right]$$

where overbars denote linear interpolation and $(\nabla p)_f$ is the face-normal gradient computed from the cell values on each side. This suppresses checkerboard modes but introduces an artificial dissipation term that depends on the mesh and timestep.

Hybrid staggered [13]: In `pimpleFoamStag`, the face-normal velocity $U_n = U \cdot \hat{n}$ is stored directly at face centres as the *primary* kinematic variable. The face flux is then exactly $\phi_f = U_n |S_f|$ —no interpolation, no Rhie-Chow correction. The pressure–velocity coupling is inherently stable because pressure and velocity unknowns live at staggered locations. The cell-centred velocity U (needed for the momentum equation, turbulence model, and post-processing) is reconstructed from U_n each iteration via a least-squares procedure:

$$U_P = T_P^{-1} \sum_f U_{n,f} S_f, \quad T_P = \sum_f \hat{n}_f \otimes S_f$$

The pressure correction step is exact at the discrete level:

$$U_n = U_n^* - \left(\frac{1}{A} \right)_f \frac{\partial p}{\partial n} \Big|_f$$

where U_n^* is the predicted face-normal velocity from the momentum equation projection and $\partial p / \partial n|_f$ is computed directly from adjacent cell pressures.

D.3 Comparison of pimpleFoamPrecise and pimpleFoamStag

Table 19 summarises the key differences between the two solvers.

Table 19: Comparison of pimpleFoamPrecise and pimpleFoamStag

Property	pimpleFoamPrecise	pimpleFoamStag
Base algorithm	Standard PIMPLE (colocated)	PIMPLE with hybrid staggered grid
Primary velocity variable	Cell-centred \mathbf{U}	Face-normal U_n
Pressure–velocity coupling	Rhie-Chow interpolation	Inherent (staggered grid)
Face flux computation	Interpolated from cell \mathbf{U}	Exact: $\phi_f = U_n S_f $
Velocity correction	$\mathbf{U} = \mathbf{u}^* - (1/A) \nabla p$	$U_n = U_n^* - (1/A)_f \partial p / \partial n$
Reconstruction step	None	Least-squares $U_n \rightarrow \mathbf{U}$ each iteration
Checkerboard risk	Suppressed by Rhie-Chow	Eliminated
Discrete mass conservation	Approximate (Rhie-Chow)	Exact
HbyA registration	Yes (persistent field)	Yes (persistent field)
Ghost cell support	Yes (explicit overwrite)	Yes (explicit overwrite)
Computational overhead	Negligible vs standard pimpleFoam	Small: one reconstruction per iteration
When to use		
Extraction phase	Recommended for -preciseBC	When staggered accuracy is needed
Reconstruction phase	Use standard pimpleFoam	Use when exact mass conservation is critical
Ghost cell mode	Full support	Full support

✓ Tip

For most users, pimpleFoamPrecise is the recommended choice for the extraction phase (it enables -preciseBC with minimal overhead). The reconstruction phase can use standard pimpleFoam. Use pimpleFoamStag when exact discrete mass conservation or elimination of Rhie-Chow artefacts is important, for example in acoustic simulations or highly under-resolved LES.

D.4 Ghost Cell Support

Both pimpleFoamStag and pimpleFoamPrecise automatically detect ghost cell mode when a ghostCells cellZone exists in the mesh. When detected, the solvers:

1. Overwrite ghost cell values for \mathbf{U} and p from extracted data at the **start** of each PIMPLE iteration (before momentum assembly)

2. Overwrite again **after** the pressure correction loop (to prevent drift from equation solving)

Ghost cell data is read from `constant/boundaryData/ghostCells/<timeName>/U` and `p`. The solvers handle `FoamFile` headers automatically.

i Note

The explicit ghost cell overwrite in the solver is an alternative to the `ghostCellConstraint` `fvOption`. The solver-side overwrite applies at the field level (outside the equation system), while the `fvOption` applies at the matrix level via `eqn.setValues()`. Both achieve the same result; the solver-side overwrite is simpler and requires no additional configuration.

E The ghostCellConstraint fvOption

The `ghostCellConstraint` is an `fvOption` that reads time-varying ghost cell data from disk and overrides the equation system in ghost cells using `eqn.setValues()`. It is instantiated as `scalarGhostCellConstraint` and `vectorGhostCellConstraint`.

E.1 Configuration

```

1 // In constant/fvOptions
2 ghostU
3 {
4     type            vectorGhostCellConstraint;
5     selectionMode    cellZone;
6     cellZone         ghostCells;
7     dataDir          "constant/boundaryData";
8     fields           (U);
9 }
10
11 ghostP
12 {
13     type            scalarGhostCellConstraint;
14     selectionMode    cellZone;
15     cellZone         ghostCells;
16     dataDir          "constant/boundaryData";
17     fields           (p);
18 }
```

Listing 50: `ghostCellConstraint` `fvOption` configuration

E.2 How It Works

At each timestep, the `fvOption`:

1. Reads ghost cell values from `<dataDir>/ghostCells/<timeName>/<fieldName>`
2. Maps data indices to cell indices using `ghostCellMap` (written by `spaceTimeWindowInitCase` with the `-ghostCells` flag)

Table 20: ghostCellConstraint parameters

Parameter	Type	Required	Description
cellZone	word	Yes	Name of the ghost cell zone
dataDir	fileName	No	Path to boundaryData directory (default: constant/boundaryData)
fields	wordList	Yes	Fields to constrain

3. Calls `eqn.setValues(cells, values)` to override the matrix rows for ghost cells

The `setValues()` call replaces the matrix row for each ghost cell with a unit diagonal and moves the prescribed value to the source term, ensuring the linear solver produces exactly the prescribed value regardless of neighbouring cell contributions.

i Note

For the pressure equation, the solver must call `fvOptions.constrain(pEqn)` after assembling the pressure matrix and before solving. Both `pimpleFoamStag` and `pimpleFoamPrecise` include this call. Standard `pimpleFoam` does not—if using the `fvOption` approach with standard `pimpleFoam`, add `fvOptions.constrain(pEqn)` to `pEqn.H`.

References

- [1] ERCOFTAC. Classic collection database: Case 043 – flow around a square cylinder. <http://cfd.mace.manchester.ac.uk/ercoftac/>, 2026.
- [2] Alin-Adrian Anton. *Space-Time Window Reconstruction in High-Performance Numeric Simulations: Application for CFD*. PhD thesis, Universitatea Politehnica Timișoara, Timișoara, Romania, November 2011. URL <https://dspace.upt.ro/jspui/handle/123456789/643>.
- [3] Alin-Adrian Anton. A method for submodelling inside transient flows. In *2011 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 75–82, 2011. doi:[10.1109/SYNASC.2011.48](https://doi.org/10.1109/SYNASC.2011.48).
- [4] Alin-Adrian Anton, Vladimir Cretu, A. Ruprecht, and Sebastian Muntean. Traffic replay compression (trc): a highly efficient method for handling parallel numerical simulation data. *Proceedings of the Romanian Academy - Series A: Mathematics, Physics, Technical Sciences, Information Science*, 14:385–392, 11 2013.
- [5] Hrvoje Jasak. *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. PhD thesis, Imperial College of Science, Technology and Medicine, London, 1996. URL <http://hdl.handle.net/10044/1/8335>.
- [6] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and Matlab*, volume 113 of *Fluid Mechanics and Its Applications*. Springer, 2016. ISBN 978-3-319-16873-9. doi:[10.1007/978-3-319-16874-6](https://doi.org/10.1007/978-3-319-16874-6).

-
- [7] Joel H. Ferziger, Milovan Perić, and Robert L. Street. *Computational Methods for Fluid Dynamics*. Springer, 4th edition, 2020. ISBN 978-3-319-99691-2. doi:[10.1007/978-3-319-99693-6](https://doi.org/10.1007/978-3-319-99693-6).
 - [8] Xiang Wu, Tamer A. Zaki, and Charles Meneveau. High-fidelity inflow conditions for internal flow calculations using the boundary-layer/reynolds-averaged Navier-Stokes approach. *Physical Review Fluids*, 5:064607, 2020. doi:[10.1103/PhysRevFluids.5.064607](https://doi.org/10.1103/PhysRevFluids.5.064607).
 - [9] ERCOFTAC. Classic collection database: Case 043 – flow around a square cylinder. <http://cfd.mace.manchester.ac.uk/ercoftac/>, 2011.
 - [10] Niklas Nordin. Benchmark ercoftac ufr2-02. https://openfoamwiki.net/index.php?title=Benchmark_ercoftac_ufr2-02, 2010.
 - [11] D. A. Lyn, S. Einav, W. Rodi, and J.-H. Park. A laser-Doppler velocimetry study of ensemble-averaged characteristics of the turbulent near wake of a square cylinder. *Journal of Fluid Mechanics*, 304:285–319, 1995. doi:[10.1017/S0022112095004435](https://doi.org/10.1017/S0022112095004435).
 - [12] Olivier Petit, Alin Bosioc, Håkan Nilsson, Sebastian Muntean, and Romeo Susan-Resiga. Unsteady simulation of the flow in a swirl generator using OpenFOAM. *International Journal of Fluid Machinery and Systems*, 4(1):199–207, January–March 2011. doi:[10.5293/IJFMS.2011.4.1.199](https://doi.org/10.5293/IJFMS.2011.4.1.199).
 - [13] Krishnan Mahesh, George Constantinescu, and Parviz Moin. A numerical method for large-eddy simulation in complex geometries. *Journal of Computational Physics*, 197(1): 215–240, 2004. doi:[10.1016/j.jcp.2003.11.031](https://doi.org/10.1016/j.jcp.2003.11.031).
 - [14] C. M. Rhie and W. L. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532, 1983. doi:[10.2514/3.8284](https://doi.org/10.2514/3.8284).