spaceTimeWindow Library

Technical Manual

Version 1.0 (Draft) | OpenFOAM v2512

Boundary Data Extraction • **Time Interpolation** • **Optional Encryption**

Alin-Adrian Anton

Politehnica University of Timişoara

Last updated: February 3, 2026

Contents

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
- 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: libsodium for encryption support
- Optional: MPI for parallel execution

OPENFOAM® is a registered trademark of OpenCFD Limited.

1.2 Building the Library

```
# Navigate to source directory
cd openfoam-spaceTimeWindow

# Build library and utilities
./Allwmake

# Or with encryption support (requires libsodium)
./Allwmake  # Auto-detects libsodium if installed
```

Listing 1: Building spaceTimeWindow

For manual building with encryption:

```
cd src/spaceTimeWindow
export FOAM_USE_SODIUM=1 && wmake

cd applications/utilities/preProcessing/spaceTimeWindowInitCase
export FOAM_USE_SODIUM=1 && wmake

cd ../spaceTimeWindowKeygen
export FOAM_USE_SODIUM=1 && wmake
```

Listing 2: Manual build with encryption

2 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

2.1 Key Generation

Generate a keypair using the provided utility:

```
spaceTimeWindowKeygen
# Output:
# Public key:  fqzYQ0U8j27tFEr5WzEMylbvXYP+9CAyk0JhwwZ2rwg=
# Private key: QgzxB5b+DGPQH8exbWDe18n4Kv0nu5gqljI2RPBCwl4=
#
# 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.

2.2 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.

2.3 Encrypted Extraction

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

```
functions
{
    extractSubset
    {
        type            spaceTimeWindowExtract;
        libs            (spaceTimeWindow);

        box             ((0.05 -0.25 0.01) (0.90 0.25 0.38));
        outputDir       "../subset-case";
        fields          (U p nut);

        writeFormat     deltaVarint;

        // Enable encryption with public key
        publicKey       "fqzYQ0U8j27tFEr5WzEMylbvXYP+9CAyk0JhwwZ2rwg=";

        writeControl    timeStep;
        writeInterval   1;
    }
}
```

Listing 4: Encrypted extraction configuration

Encrypted files have the `.enc` extension (e.g., `U.dvz.enc` for compressed and encrypted data).

2.4 Decryption During Case Initialization

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

```
cd subset-case
spaceTimeWindowInitCase -sourceCase ../source-case
# 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

2.5 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

2.5.1 Cryptographic Foundations

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

$$P_{\text{pub}} = s \cdot G \tag{1}$$

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:

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

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

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

where M is the plaintext boundary data and $\|$ 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)):

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

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

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

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

2.6 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.

2.7 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.

2.7.1 Secure Archival Strategy

The reconstruction case contains only:

- The subset mesh (`constant/polyMesh`)
- Encrypted boundary data (`constant/boundaryData/*.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.

2.7.2 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.

3 Boundary Condition Approaches

The library provides two approaches for applying boundary conditions on the extraction boundary (`oldInternalFaces`):

3.1 Inlet-Outlet BC (`-inletOutletBC`) — Recommended for Unsteady Flows

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 **physically correct** for unsteady turbulent flows because:

- Turbulent flows have instantaneous velocity fluctuations that can reverse direction locally
- Vortex shedding, recirculation zones, and turbulent eddies cause portions of the boundary to alternate between inflow and outflow

- Prescribing fixed values at outflow faces is non-physical (information should leave the domain freely)
- The flux-based switching automatically adapts to the instantaneous flow direction at each face

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

Listing 6: Using inlet-outlet BC

3.2 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.

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

Listing 7: Using fixed outlet direction

△ Warning

These options are mutually exclusive. Using both `-inletOutletBC` and `-outletDirection` together produces an error with guidance.

3.3 Boundary Condition Assignment Logic

For each field in the initial time directory, `spaceTimeWindowInitCase` assigns BCs according to Table 1.

Table 1: Boundary condition assignment logic

In boundaryData?	-inletOutletBC?	Field Type	BC Applied
Yes	Yes	vector (U)	`spaceTimeWindowInletOutlet`
Yes	Yes	scalar	`zeroGradient`
Yes	No	any	`spaceTimeWindow` (Dirichlet)
No	any	any	`zeroGradient`

This ensures fields without time-varying data automatically get appropriate BCs.

4 Workflow Overview

The space-time window reconstruction workflow is strictly sequential, as shown in fig. 1.

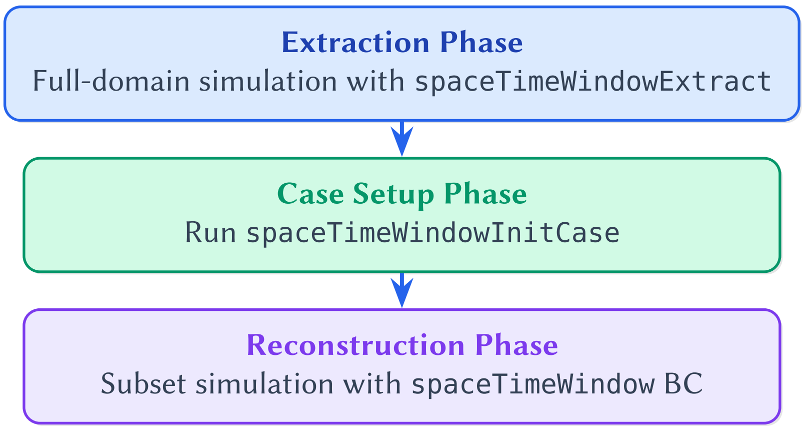
Extraction Phase
Full-domain simulation with spaceTimeWindowExtract
Case Setup Phase
Run spaceTimeWindowInitCase
Reconstruction Phase
Subset simulation with spaceTimeWindow BC

Figure 1: Space-time window reconstruction workflow

4.1 Serial Workflow

```
# 1. Run extraction during simulation
cd source-case
pimpleFoam    # With spaceTimeWindowExtract function object

# 2. Initialize reconstruction case (recommended: inlet-outlet BC)
cd ../subset-case
spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC

# 3. Run reconstruction
pimpleFoam
```

Listing 8: 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.

4.2 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)

```
# Step 1: Run parallel extraction
cd source-case
mpirun -np 4 pimpleFoam -parallel
# Extraction automatically forces writes at t_0, t_1, t_2

# Step 2: Reconstruct fields at cubic-safe start time (t_2)
reconstructPar -time 0.0002

# Step 3: Initialize subset case (recommended: inlet-outlet BC)
cd ../subset-case
spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC

# Step 4: Run reconstruction
pimpleFoam
```

Listing 9: 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

5 Configuration Reference

5.1 Extraction Configuration

The extraction function object is configured in `system/controlDict` within the `functions` sub-dictionary. The available parameters are listed in table 2.

```
functions
{
    extractSubset
    {
        type            spaceTimeWindowExtract;
        libs            (spaceTimeWindow);

        // Bounding box: ((minX minY minZ) (maxX maxY maxZ))
        // Must be fully internal to domain
        box             ((0.05 -0.25 0.01) (0.90 0.25 0.38));

        outputDir       "../subset-case";

        // Fields for time-varying boundary data (written every timestep)
        fields          (U);             // Typically just velocity

        // Fields for initial conditions (optional, defaults to 'fields')
        initialFields   (U p nut);        // More fields for IC

        // Write format: ascii, binary, deltaVarint, or dvzt
        writeFormat     dvzt;             // Recommended: best compression

        // Precision for deltaVarint/dvzt (default: 6)
        deltaVarintPrecision  6;

        // Keyframe interval for dvzt (default: 20)
        dvztKeyframeInterval  20;

        // Gzip compression (ignored for deltaVarint/dvzt)
        writeCompression off;

        // Optional encryption
        // publicKey     "base64-encoded-public-key";

        // Time window (optional)
        // timeStart     0.0;
        // timeEnd       1.0;

        writeControl    timeStep;
        writeInterval   1;
    }
}
```

Listing 10: Complete extraction configuration

Table 2: 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 (boundaryData)
`initialFields`	wordList	No	Fields for initial conditions (default: same as fields)
`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

5.2 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

This allows extracting more fields for initial conditions (e.g., `U p nut k omega`) while only storing time-varying data for velocity:

```
// In extraction function object
fields          (U);               // Only U for time-varying BC (saves
    storage)
initialFields   (U p nut);         // More fields for initial conditions
```

Listing 11: Field selection strategy

Fields NOT in `boundaryData` automatically get `zeroGradient` BC on `oldInternalFaces`.

✓ Tip

For unsteady turbulent flows with `-inletOutletBC`, the recommended approach is to extract only velocity (U) for boundary data while including all turbulence fields for initial conditions. This works because pressure and turbulence quantities use zeroGradient (no BC data needed), reducing storage by approximately 66%.

5.3 Boundary Condition Configuration

5.3.1 spaceTimeWindowInletOutlet (Recommended)

Flux-based boundary condition that reads pre-computed velocity values and applies them only at inflow faces. The parameters are listed in table 3.

```
boundaryField
{
    oldInternalFaces
    {
        type                    spaceTimeWindowInletOutlet;
        dataDir                 "constant/boundaryData";
        phi                     phi;          // Flux field name
        allowTimeInterpolation  true;
        timeInterpolationScheme cubic;        // or "linear" or "none"
        value                   uniform (0 0 0);
    }
}
```

Listing 12: 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 3: 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`

5.3.2 spaceTimeWindow (Pure Dirichlet)

Pure Dirichlet boundary condition that prescribes values on all faces regardless of flow direction. The parameters are listed in table 4.

```
boundaryField
{
    oldInternalFaces
    {
        type                    spaceTimeWindow;
        dataDir                 "constant/boundaryData";
        fixesValue              true;         // Tells adjustPhi these values
            are fixed
        allowTimeInterpolation  true;
        timeInterpolationScheme cubic;
        reportFlux              true;
        value                   uniform (0 0 0);
    }
}
```

Listing 13: spaceTimeWindow configuration

Use this for:

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

5.4 Case Initialization Options

The `spaceTimeWindowInitCase` utility accepts command-line options (see table 5):

```
# Recommended: inlet-outlet BC for unsteady turbulent flows
spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC

# Alternative: fixed outlet direction for steady-mean flows
spaceTimeWindowInitCase -sourceCase ../source-case -outletDirection "(1 0 0)"
```

Table 4: 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

```

# With mass flux correction (optional, ensures exact mass conservation)
spaceTimeWindowInitCase -sourceCase ../source-case -inletOutletBC
    -correctMassFlux
```

Listing 14: spaceTimeWindowInitCase usage

Table 5: spaceTimeWindowInitCase options

Option	Type	Description
`-sourceCase`	directory	Source case where extraction ran (required)
`-extractDir`	directory	Directory with extracted data (default: cwd)
`-inletOutletBC`	flag	**Recommended.** Use flux-based inlet-outlet BC for U, zeroGradient for scalars
`-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)
`-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**

`-inletOutletBC` and `-outletDirection` are mutually exclusive. Using both together produces an error with guidance on which option to choose.

5.5 Mesh Coarsening and Refinement

The reconstruction mesh can be coarsened or refined relative to the extraction mesh (table 6). This enables running reconstructions at different resolutions than the original simulation.

```
# Refine mesh (finer than extraction)
spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -refineLevel 1

# Coarsen mesh (coarser than extraction)
spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -coarsenLevel 1

# Multiple levels
spaceTimeWindowInitCase -sourceCase ../source -inletOutletBC -refineLevel 2
```

Listing 15: Mesh resolution options

Table 6: Mesh resolution options

Option	Type	Description
`-refineLevel`	label	Refine mesh N times (each level splits cells ×8)
`-coarsenLevel`	label	Coarsen mesh N times (each level merges cells)

△ Warning

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

5.5.1 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. 2. 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 $\boldsymbol{p}$ inside a triangle with vertices $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$, the barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$ satisfy eq. (6):

$$\boldsymbol{p} = \lambda_1 \boldsymbol{v}_1 + \lambda_2 \boldsymbol{v}_2 + \lambda_3 \boldsymbol{v}_3, \quad \text{where} \quad \lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{6}$$

The weights are computed from signed triangle areas (eq. (7)):

$$\lambda_1 = \frac{A(\boldsymbol{p}, \boldsymbol{v}_2, \boldsymbol{v}_3)}{A(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)}, \quad \lambda_2 = \frac{A(\boldsymbol{v}_1, \boldsymbol{p}, \boldsymbol{v}_3)}{A(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)}, \quad \lambda_3 = \frac{A(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{p})}{A(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)} \tag{7}$$

where the signed area of a 2D triangle is given by eq. (8):

$$A(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = \frac{1}{2}\left[(b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)\right] \tag{8}$$

The interpolated field value at the target point is then (eq. (9)):

$$\phi(\boldsymbol{p}) = \lambda_1 \phi_1 + \lambda_2 \phi_2 + \lambda_3 \phi_3 \tag{9}$$

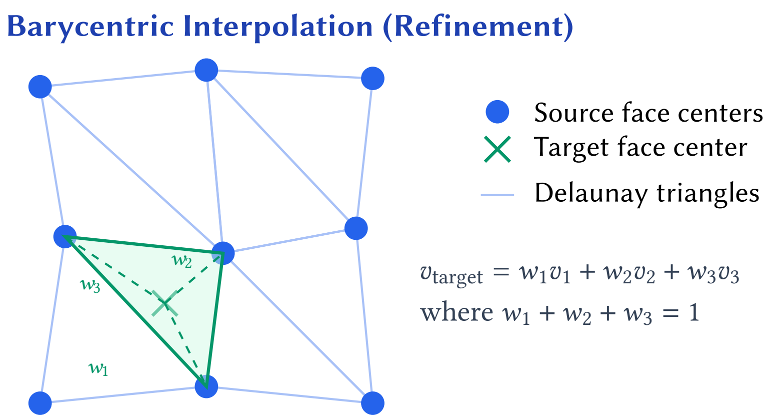
Barycentric Interpolation (Refinement)
Source face centers
Target face center
Delaunay triangles
w_2
w_3
w_1
$v_{\text{target}} = w_1 v_1 + w_2 v_2 + w_3 v_3$
where $w_1 + w_2 + w_3 = 1$

Figure 2: 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. 3. 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 $\boldsymbol{p}$ with search radius r, define the set of contributing source points (eq. (10)):

$$\mathcal{S}(\boldsymbol{p}, r) = \{i : \|\boldsymbol{x}_i - \boldsymbol{p}\| \leq r\} \tag{10}$$

The interpolated value is the arithmetic mean of all contributing sources (eq. (11)):

$$\phi(\boldsymbol{p}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \phi_i \tag{11}$$

If $|\mathcal{S}| = 0$, the search radius is expanded by a factor $\alpha > 1$ (typically $\alpha = 1.5$) and the search is repeated (eq. (12)):

$$r \leftarrow \alpha \cdot r \quad \text{until} \quad |\mathcal{S}(\boldsymbol{p}, r)| > 0 \tag{12}$$

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

2D Projection by Box Face The interpolation is performed on 2D point clouds, as illustrated in fig. 4. 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.

5.5.2 Initial Field Interpolation

Initial fields are also interpolated when mesh resolution changes (see figs. 5 and 6):

- **Refinement**: Uses `mapFields` with cell-center interpolation (fig. 5)
- **Coarsening**: Uses volume-weighted averaging of source cells (fig. 6)

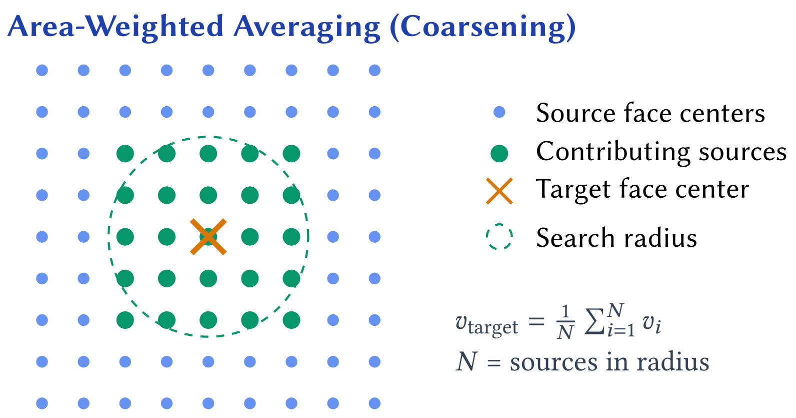

Figure 3: Area-weighted averaging for mesh coarsening: all source face centers (blue) within the search radius of each target point (orange) are averaged with equal weights

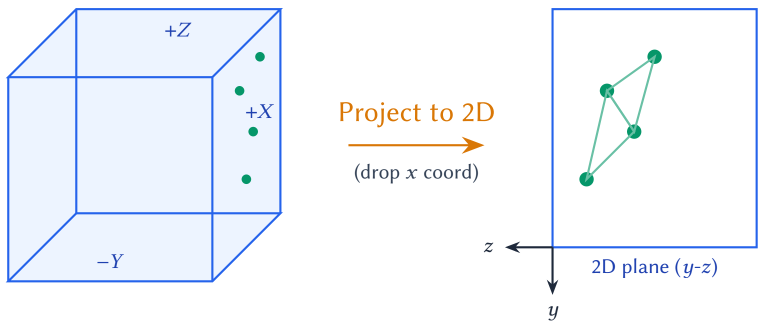

Figure 4: Points on each box face are projected to 2D for triangulation. The constant coordinate (perpendicular to the face) is dropped, enabling efficient 2D algorithms

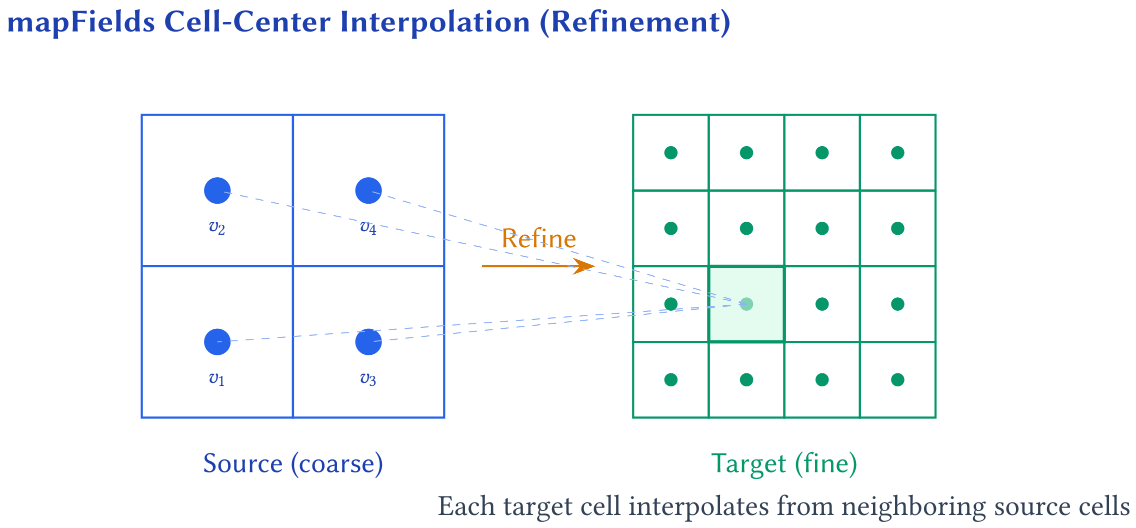
mapFields Cell-Center Interpolation (Refinement)
v_2
v_4
v_1
v_3
Refine
Source (coarse)
Target (fine)
Each target cell interpolates from neighboring source cells

Figure 5: Initial field refinement using mapFields: target cell values are interpolated from surrounding source cell centers using distance-weighted averaging

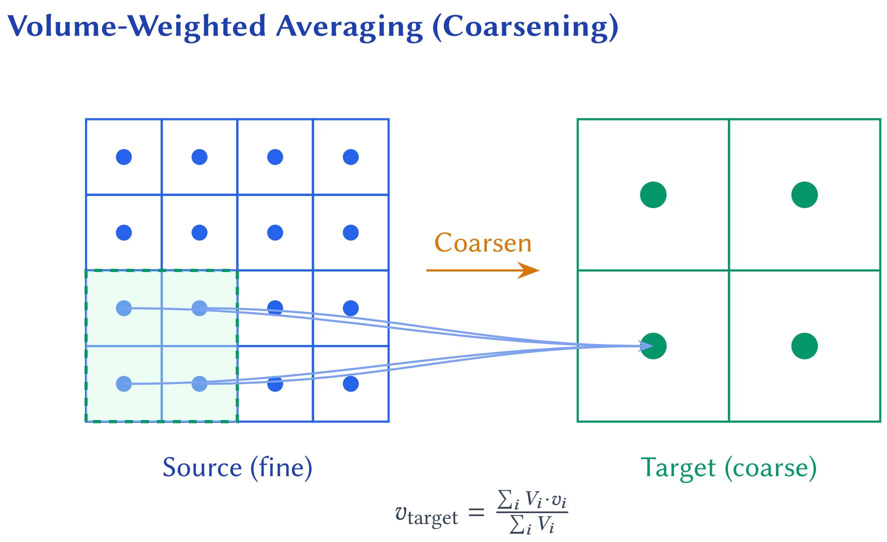
Volume-Weighted Averaging (Coarsening)
Coarsen
Source (fine)
Target (coarse)
$v_{\text{target}} = \frac{\sum_i V_i \cdot v_i}{\sum_i V_i}$

Figure 6: Initial field coarsening using volume-weighted averaging: source cell values within each target cell region are averaged weighted by their volumes

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:

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

6 Data Storage Format

6.1 Directory Structure

The extraction creates the following directory structure:

```
outputDir/
    constant/
        polyMesh/               # Subset mesh
            points
            faces
            owner
            neighbour
            boundary
            extractionBox       # (parallel only)
        boundaryData/
            oldInternalFaces/
                points              # Face centres
                extractionMetadata  # Settings and timestep list
                0.0001/
                    U               # or U.dvz or U.dvz.enc
                    p
                    nut
                0.0002/
                    ...
    0.0001/                     # Initial fields (serial only)
        U
        p
        nut
```

Listing 16: Extracted data structure

6.2 Boundary Data Compression

The `writeFormat` parameter controls how boundary data files are written, with compression ratios summarized in table 7.

Table 7: 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
dvzt	.dvzt	~2.4%	Best compression, recommended

6.2.1 Delta-Varint Codec (DVZ)

Specialized codec optimized for CFD boundary data (see figs. 7 and 8):

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

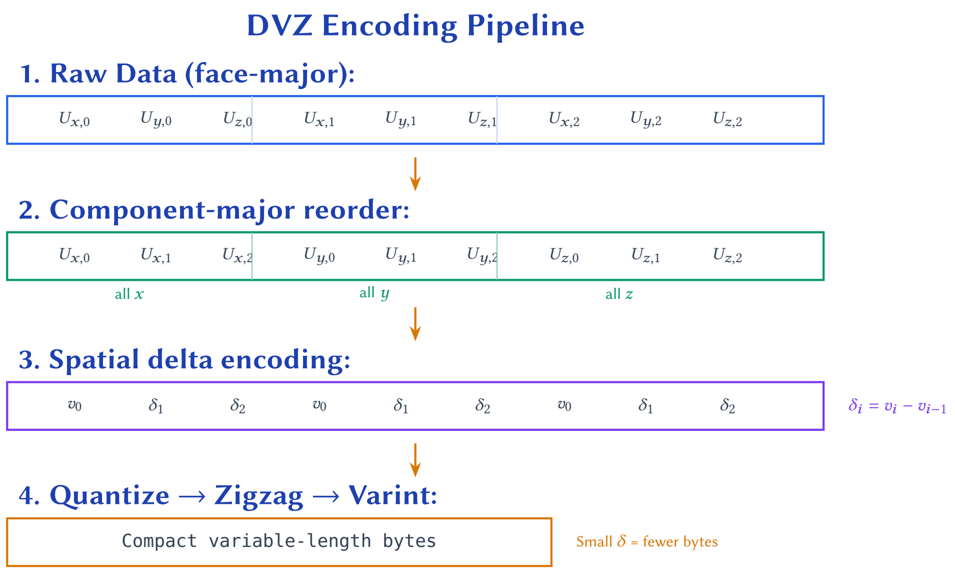
DVZ Encoding Pipeline
1. Raw Data (face-major):
$U_{x,0}$ $U_{y,0}$ $U_{z,0}$ $U_{x,1}$ $U_{y,1}$ $U_{z,1}$ $U_{x,2}$ $U_{y,2}$ $U_{z,2}$
2. Component-major reorder:
$U_{x,0}$ $U_{x,1}$ $U_{x,2}$ $U_{y,0}$ $U_{y,1}$ $U_{y,2}$ $U_{z,0}$ $U_{z,1}$ $U_{z,2}$
all x
all y
all z
3. Spatial delta encoding:
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 7: 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

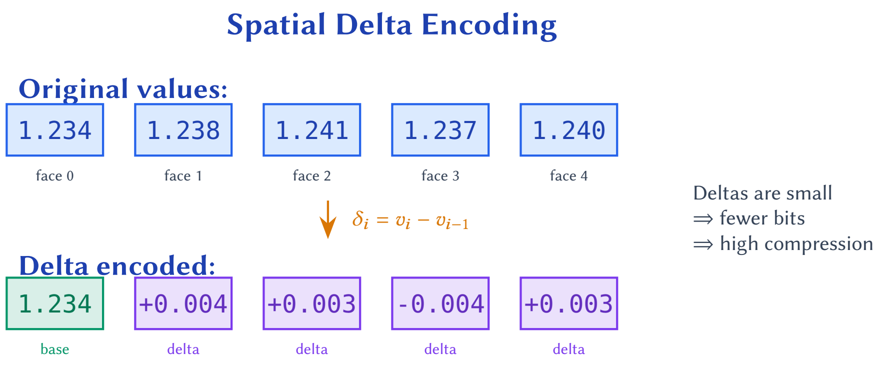

Figure 8: Spatial delta encoding: instead of storing absolute values, DVZ stores the first value and differences between consecutive faces. Smooth CFD fields have small deltas, enabling high compression

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.

6.2.2 DVZ Mathematical Formulation

Component-Major Reordering For a vector field $\boldsymbol{U}$ with N faces, the raw data layout is face-major (eq. (13)):

$$\text{Raw} : [U_{x,0}, U_{y,0}, U_{z,0}, U_{x,1}, U_{y,1}, U_{z,1}, \ldots, U_{x,N-1}, U_{y,N-1}, U_{z,N-1}] \tag{13}$$

DVZ reorders to component-major for better compression (eq. (14)):

$$\text{Reordered} : [\underbrace{U_{x,0}, U_{x,1}, \ldots, U_{x,N-1}}_{\text{all } x}, \underbrace{U_{y,0}, \ldots, U_{y,N-1}}_{\text{all } y}, \underbrace{U_{z,0}, \ldots, U_{z,N-1}}_{\text{all } z}] \tag{14}$$

Spatial Delta Encoding For each component sequence $\{v_0, v_1, \ldots, v_{N-1}\}$, store deltas as shown in eq. (15):

$$\delta_0 = v_0, \quad \delta_i = v_i - v_{i-1} \quad \text{for } i = 1, \ldots, N-1 \tag{15}$$

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. (16):

$$q_i = \text{round}(\delta_i \times 10^p) \tag{16}$$

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. (17)):

$$z_i = \begin{cases} 2q_i & \text{if } q_i \geq 0 \\ -2q_i - 1 & \text{if } q_i < 0 \end{cases} = (q_i \ll 1) \oplus (q_i \gg 31) \tag{17}$$

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. (18)):

$$\text{bytes}(z) = \left\lceil \frac{\lfloor \log_2(z) \rfloor + 1}{7} \right\rceil \tag{18}$$

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.

6.2.3 Delta-Varint-Temporal Codec (DVZT)

Enhanced codec that exploits both spatial and temporal correlation for better compression (see figs. 9 to 11):

1. **Keyframes** (every N timesteps): Self-contained, same as DVZ (fig. 9)
2. **Delta frames**: Hybrid spatial-temporal prediction (fig. 10)
    - 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 ~10% smaller than DVZ for delta frames

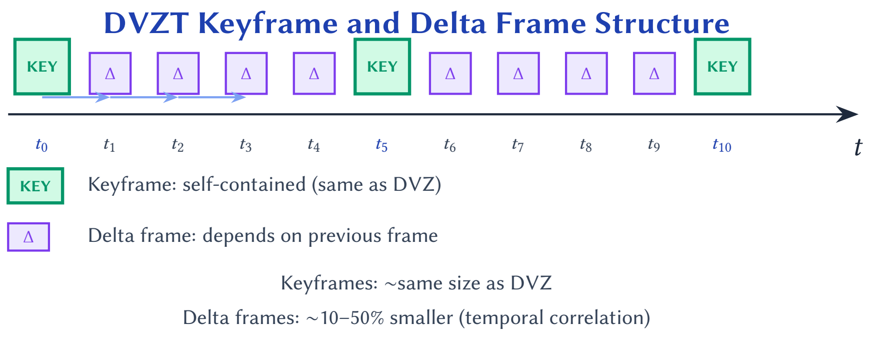
DVZT Keyframe and Delta Frame Structure
KEY
Δ
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
t
Keyframe: self-contained (same as DVZ)
Delta frame: depends on previous frame
Keyframes: ~same size as DVZ
Delta frames: ~10–50% smaller (temporal correlation)

Figure 9: DVZT frame structure: keyframes are self-contained and appear every N timesteps (configurable). Delta frames between keyframes use temporal prediction for better compression

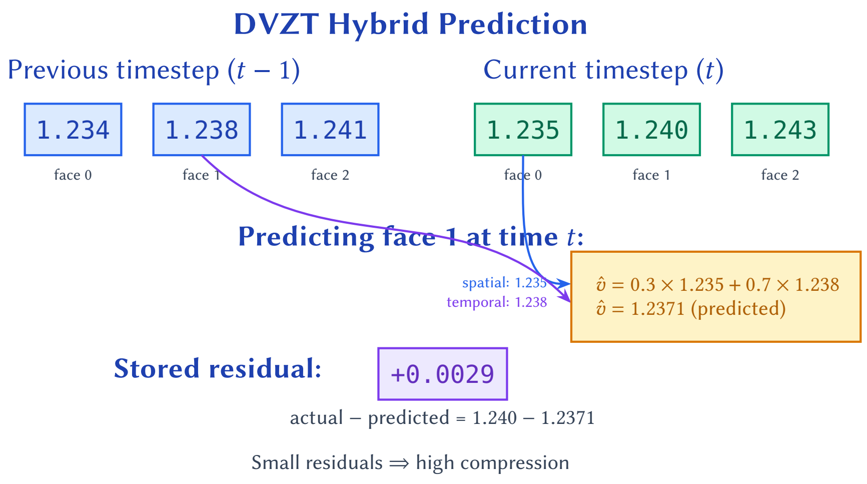
DVZT Hybrid Prediction
Previous timestep $(t-1)$
Current timestep (t)
1.234 face 0
1.238 face 1
1.241 face 2
1.235 face 0
1.240 face 1
1.243 face 2
Predicting face 1 at time t:
spatial: 1.235
temporal: 1.238
$\hat{v} = 0.3 \times 1.235 + 0.7 \times 1.238$
$\hat{v} = 1.2371$ (predicted)
Stored residual:
+0.0029
actual − predicted = 1.240 − 1.2371
Small residuals ⟹ high compression

Figure 10: 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

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

Listing 17: DVZT configuration

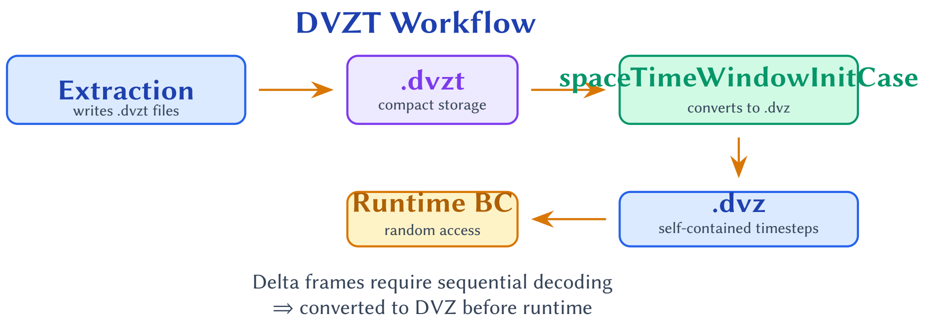

Figure 11: DVZT workflow: extraction writes compact .dvzt files. Before reconstruction, spaceTimeWindowInitCase converts them to .dvz format (required for random timestep access at runtime)

6.2.4 DVZT Mathematical Formulation

Frame Types DVZT distinguishes between keyframes and delta frames based on the timestep index k (eq. (19)):

$$\text{Frame type}(k) = \begin{cases} \text{Keyframe} & \text{if } k \mod K = 0 \\ \text{Delta frame} & \text{otherwise} \end{cases} \tag{19}$$

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

Keyframe Encoding Keyframes use pure DVZ encoding (spatial delta only), as shown in eq. (20):

$$\delta_i^{(k)} = v_i^{(k)} - v_{i-1}^{(k)} \quad \text{(spatial delta)} \tag{20}$$

Delta Frame Hybrid Prediction Delta frames use a weighted combination of spatial and temporal neighbors (eq. (21)):

$$\hat{v}_i^{(k)} = \alpha \cdot v_{i-1}^{(k)} + (1-\alpha) \cdot v_i^{(k-1)} \tag{21}$$

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

The residual to be encoded is given by eq. (22):

$$r_i^{(k)} = v_i^{(k)} - \hat{v}_i^{(k)} \tag{22}$$

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. (23)):

$$\sigma_r^2 \approx \alpha^2\sigma_s^2 + (1-\alpha)^2\sigma_t^2 \tag{23}$$

For small timesteps where $\sigma_t^2 \ll \sigma_s^2$ (eq. (24)):

$$\frac{\sigma_r}{\sigma_s} \approx \alpha = 0.3 \tag{24}$$

This explains the ~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. (25)):

$$v_i^{(k)} = r_i^{(k)} + \alpha \cdot v_{i-1}^{(k)} + (1 - \alpha) \cdot v_i^{(k-1)} \tag{25}$$

This creates a dependency chain from each keyframe (eq. (26)):

$$\text{Keyframe } k_0 \longrightarrow \text{Frame } k_0 + 1 \longrightarrow \cdots \longrightarrow \text{Frame } k_0 + K - 1 \tag{26}$$

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. During case initialization, the utility automatically converts `.dvzt` to `.dvz` format (required because delta frames need sequential processing). The resulting `.dvz` files are read by the spaceTimeWindow BC at runtime.

Extraction ⟶ `.dvzt` files (~10% smaller) ⟶ `spaceTimeWindowInitCase` ⟶ `.dvz` 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 8):

Table 8: 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

6.2.5 External Compression Benchmark

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 9 for DVZ files and table 10 for DVZT files.

Table 9: 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	
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	Best balance
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	Fastest

Table 10: 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

Key findings:

- **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**: zstd -3 (~10% ratio, 300–400 MB/s)
- **Archive/transfer**: 7z lzma2 -mx9 or xz -6 (~6% ratio, 10 MB/s)
- **Real-time streaming**: lz4 (~14–18% ratio, 600–800 MB/s)

```
# Archive with best compression (7z)
cd subset-case/constant/boundaryData/oldInternalFaces
7z a -m0=lzma2 -mx=9 ../boundaryData.7z */U.dvz */U.dvzt

# Or with zstd for faster compression
tar -cf - */U.dvz */U.dvzt | zstd -3 > ../boundaryData.tar.zst
```

Listing 18: Archival compression example

6.3 Extraction Metadata

The `extractionMetadata` file contains:

```
{
    openfoamVersion     "v2512";
    openfoamApi         2512;
    solver              "pimpleFoam";
    deltaT              1e-04;
    adjustTimeStep      false;
    timePrecision       6;
    extractionStartTime 0.0001;
    boxMin              (0.05 -0.25 0.01);
    boxMax              (0.90 0.25 0.38);
    nGlobalFaces        12345;
    timesteps           (0.0001 0.0002 0.0003 ...);
}
```

Listing 19: extractionMetadata contents

7 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.

7.1 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

```
# Decompose the case
decomposePar

# Run parallel extraction
mpirun -np 8 pimpleFoam -parallel

# Automatic field writes occur at:
#   t_0 - extraction start (for lookback)
#   t_1 - linear interpolation start
#   t_2 - cubic interpolation start
```

Listing 20: Parallel extraction

7.2 Field Reconstruction

After parallel extraction, reconstruct fields before case initialization:

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

# For linear interpolation
reconstructPar -time <t_1>
```

Listing 21: 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.

7.3 Parallel Reconstruction

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

```
cd subset-case
decomposePar
mpirun -np 4 pimpleFoam -parallel
reconstructPar
```

Listing 22: 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.

8 Mass Conservation

8.1 The Mass Imbalance Problem

Face interpolation during extraction can introduce small mass flux imbalances:

$$\text{imbalance} = \sum_f \mathbf{U}_f \cdot \mathbf{S}_f \neq 0 \tag{27}$$

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.

8.2 Solutions

8.2.1 Use `-inletOutletBC` (Recommended)

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

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

Listing 23: Using inlet-outlet BC for mass conservation

8.2.2 Use `-correctMassFlux`

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

$$\mathbf{U}_{\text{corrected}} = \mathbf{U} - \frac{\text{imbalance}}{\sum_f |\mathbf{S}_f|} \cdot \hat{\mathbf{n}} \tag{28}$$

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:

$$\sum_f \mathbf{U}_{\text{corrected}} \cdot \mathbf{S}_f = 0 \tag{29}$$

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

Listing 24: Using mass flux correction

8.2.3 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)

8.3 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)

8.4 Outlet Patch for Pressure Relief

The `-outletDirection` option creates an outlet patch (see fig. 12):

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

Listing 25: Creating outlet patch

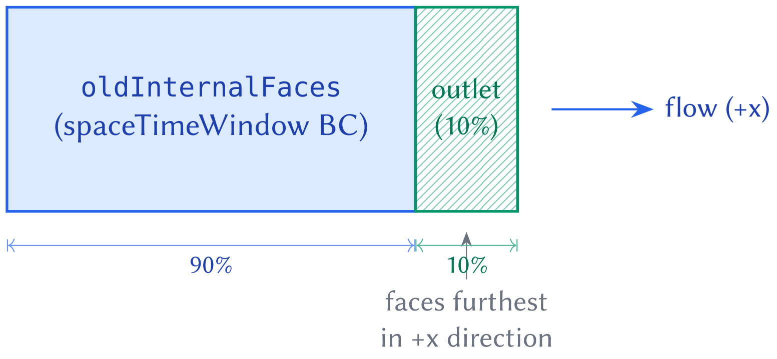

Figure 12: Outlet patch creation with `-outletDirection "(1 0 0)"`

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

8.5 Recommended Configurations

The recommended mass conservation strategies are summarized in table 11.

9 Solver Settings and Relaxation

9.1 Time Stepping

For accurate reconstruction, use identical time stepping:

Table 11: Mass conservation strategies

Configuration	Description
`-inletOutletBC`	**Recommended** for unsteady turbulent flows (vortex shedding, etc.). Natural mass balance through flux-based switching.
`-inletOutletBC -correctMassFlux`	Unsteady turbulent + extra safety. Ensures exact mass balance with inlet-outlet switching.
`-outletDirection "(1 0 0)"`	Steady-mean flow direction. Use when outlet location is known and flow direction is consistent.
`-correctMassFlux` (no outlet)	Maximum fidelity. All faces prescribed with exact mass balance.

```
deltaT          1e-04;      // Must match extraction
adjustTimeStep  no;         // Fixed timestep recommended

// If adaptive timestep is required:
adjustTimeStep  yes;
maxCo           0.5;
maxDeltaT       1e-03;
```

Listing 26: Recommended controlDict settings

△ Warning

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

9.2 PIMPLE Settings

For incompressible flows:

```
PIMPLE
{
    nOuterCorrectors    2;
    nCorrectors         2;
    nNonOrthogonalCorrectors 1;

    // Pressure reference (set by spaceTimeWindowInitCase)
    pRefPoint           (0.5 0 0.2);
    pRefValue           0;
}
```

Listing 27: Recommended PIMPLE settings

9.3 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:

```
relaxationFactors
{
    fields
    {
        p               0.7;
    }
    equations
    {
        U               0.7;
        "(k|epsilon|omega|nuTilda)"  0.7;
    }
}
```

Listing 28: Optional relaxation

9.4 Linear Solver Settings

Standard settings work well:

```
solvers
{
    p
    {
        solver          GAMG;
        smoother        GaussSeidel;
        tolerance       1e-06;
        relTol          0.01;
    }

    U
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-06;
        relTol          0.1;
    }
}
```

Listing 29: Linear solver settings

10 Time Interpolation

The boundary conditions support three time interpolation modes, summarized in Table 12 and illustrated in fig. 13. The comparison between linear and cubic interpolation quality is shown in fig. 14.

Table 12: Time interpolation modes

Mode	Start Time	Timesteps Used	Use Case
`none` (exact)	t_0	1 (exact match)	"Bit"-reproducible results
`linear`	t_1	2 (bracketing)	Simple, smoothly varying flows
`cubic`	t_2	4 (Catmull-Rom)	Unsteady turbulent flows (recommended)

`spaceTimeWindowInitCase` automatically sets `startTime` and `endTime` to ensure sufficient buffer timesteps for the selected interpolation scheme (see fig. 15 for buffer requirements).

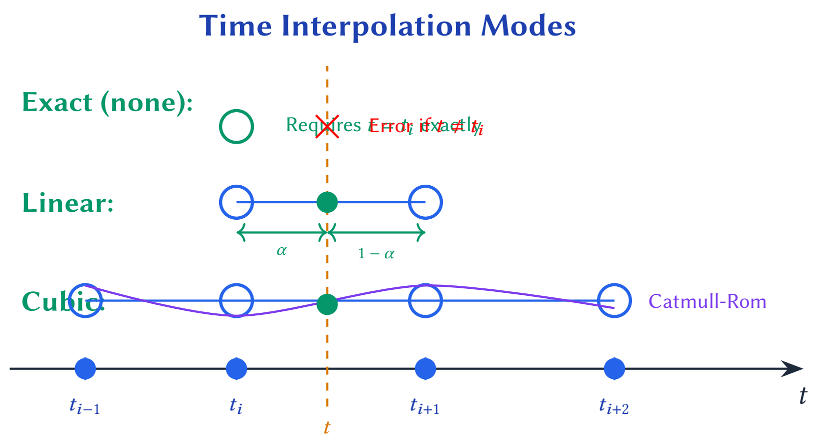

Figure 13: Time interpolation modes: exact matching uses only one timestep, linear interpolation uses two bracketing timesteps, and cubic (Catmull-Rom) uses four timesteps for smooth C^1 continuous interpolation

10.1 Exact Timestep Matching (Default)

By default, the boundary condition requires exact timestep matching:

- Simulation time must match available sample times (within 1% of `deltaT`)
- Fatal error if no matching timestep found
- Ensures "bit"-reproducible results (highest fidelity)

10.2 Linear Interpolation

For cases where exact matching is not possible:

```
oldInternalFaces
{
    type                    spaceTimeWindow;
    allowTimeInterpolation  true;
```

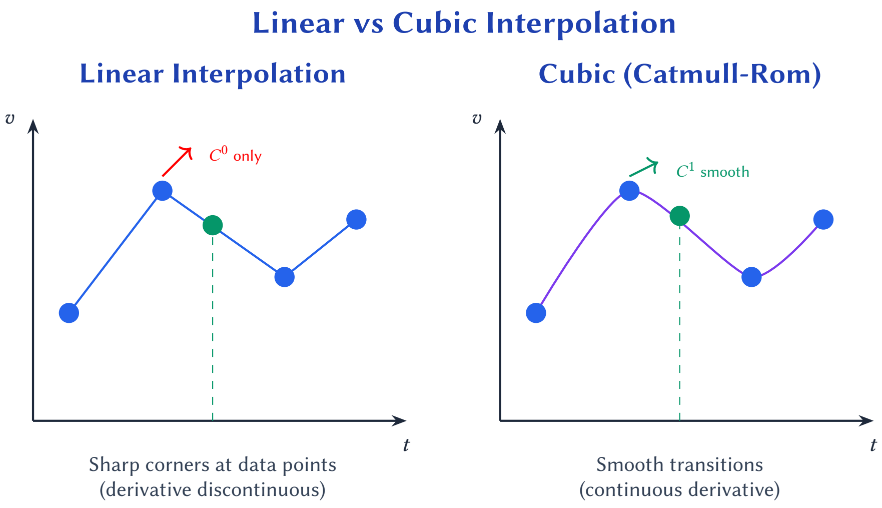

Figure 14: Comparison of linear and cubic interpolation: linear interpolation creates C^0 continuous curves with sharp corners at data points, while Catmull-Rom cubic splines provide C^1 continuity with smooth transitions

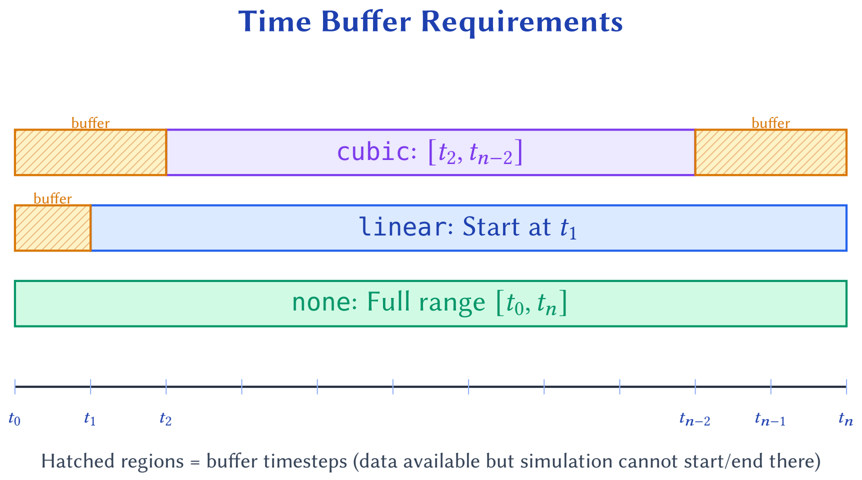

Figure 15: Time buffer requirements for each interpolation mode: exact matching can use the full range, linear needs one buffer timestep at the start, and cubic needs two buffer timesteps at both start and end

```
    timeInterpolationScheme linear;
    // ...
}
```

Listing 30: Linear interpolation

Uses two bracketing timesteps (eq. (30)):

$$\text{value} = (1-\alpha)\cdot v_i + \alpha\cdot v_{i+1} \quad \text{where} \quad \alpha = \frac{t-t_i}{t_{i+1}-t_i} \tag{30}$$

The interpolation parameter $\alpha \in [0, 1]$ represents the normalized position between the two samples. This is equivalent to first-order polynomial interpolation (eq. (31)):

$$\phi(t) = \phi_i + \frac{\phi_{i+1}-\phi_i}{t_{i+1}-t_i}(t-t_i) \tag{31}$$

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

10.3 Cubic Spline Interpolation

For smoother results with adaptive timestepping:

```
oldInternalFaces
{
    type                    spaceTimeWindow;
    allowTimeInterpolation  true;
    timeInterpolationScheme cubic;
    // ...
}
```

Listing 31: 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`

10.3.1 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. (32)):

$$d_j = |t_{j+1}-t_j|^{0.5}, \qquad j = 0, 1, 2 \tag{32}$$

The normalized parameter within the segment $[t_1, t_2]$ is (eq. (33)):

$$u = \frac{t-t_1}{t_2-t_1} \tag{33}$$

The spline is constructed using the Barry-Goldman algorithm. Define intermediate points (eq. (34)):

$$\begin{aligned} A_1 &= \frac{t_2 - t}{t_2 - t_0}\phi_0 + \frac{t - t_0}{t_2 - t_0}\phi_1 \\ A_2 &= \frac{t_2 - t}{t_2 - t_1}\phi_1 + \frac{t - t_1}{t_2 - t_1}\phi_2 \\ A_3 &= \frac{t_3 - t}{t_3 - t_1}\phi_2 + \frac{t - t_1}{t_3 - t_1}\phi_3 \end{aligned} \tag{34}$$

Then compute (eq. (35)):

$$\begin{aligned} B_1 &= \frac{t_2 - t}{t_2 - t_0}A_1 + \frac{t - t_0}{t_2 - t_0}A_2 \\ B_2 &= \frac{t_3 - t}{t_3 - t_1}A_2 + \frac{t - t_1}{t_3 - t_1}A_3 \end{aligned} \tag{35}$$

Finally, the interpolated value is (eq. (36)):

$$\phi(t) = \frac{t_2 - t}{t_2 - t_1}B_1 + \frac{t - t_1}{t_2 - t_1}B_2 \tag{36}$$

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

10.4 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.

11 Flux Reporting and Diagnostics

Enable flux reporting to monitor mass conservation:

```
oldInternalFaces
{
    type            spaceTimeWindow;
    reportFlux      true;
    // ...
}
```

Listing 32: Enabling flux reporting

Output format (fields explained in table 13):

```
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 33: Flux report output

Table 13: Flux report fields

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

12 Troubleshooting

12.1 Common Errors

12.1.1 Time Step Mismatch

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

Listing 34: deltaT mismatch error

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

12.1.2 No Matching Timestep

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

Listing 35: Missing timestep error

Solution: Enable time interpolation with `allowTimeInterpolation true`.

12.1.3 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

12.1.4 Encryption Errors

```
FOAM FATAL ERROR:
Failed to decrypt file: constant/boundaryData/.../U.dvz.enc
```

Listing 36: 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

13 Acknowledgments

This library implements the space-time window reconstruction method developed in [1]. The example case uses mesh generation from the ERCOFTAC Classic Collection Database [2].

A Example Case: ERCOFTAC UFR2-02 Square Cylinder

The library includes a complete example based on the ERCOFTAC UFR2-02 benchmark case [2, 3] (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.

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. [3] (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**: $\mathrm{St} = fD/U_\infty \approx 0.13$ for square cylinders, where f is the shedding frequency, D is the cylinder side length, and U_∞ is the freestream velocity
- **Turbulent wake**: At $\mathrm{Re} = 21{,}400$, 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. 16) 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.

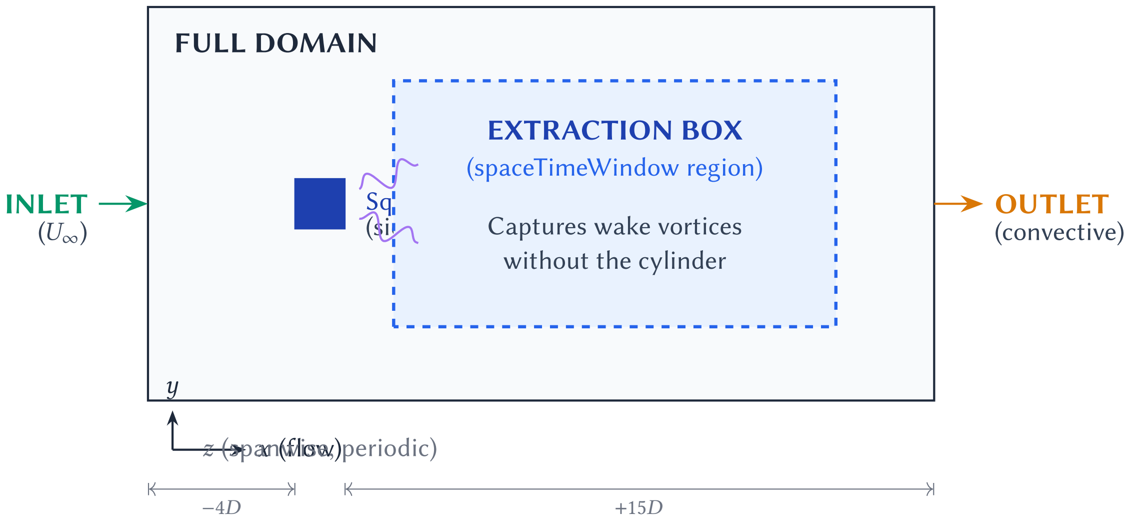

Figure 16: Computational domain for the square cylinder case showing the extraction box placement in the wake region

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:

```
extractSubset
{
    type            spaceTimeWindowExtract;
    libs            (spaceTimeWindow);

    // Box starts 0.5D downstream of cylinder, extends to 9D
    // Lateral extent captures the wake width
    // Full spanwise extent (periodic direction)
    box             ((0.05 -0.25 0.01) (0.90 0.25 0.38));

    outputDir       "../cylinder-subset";
    fields          (U p nut);
    writeFormat     deltaVarint;

    writeControl    timeStep;
    writeInterval   1;
}
```

Listing 37: Extraction configuration for square cylinder

A.5 Physical Parameters

The physical parameters for this case are listed in table 14.

Table 14: Square cylinder case parameters

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

A.6 Running the Example

```
cd examples/ufr2-02

# Generate mesh and run full simulation
./Allrun

# The extraction creates ../ufr2-02-subset with:
#   - Subset mesh in constant/polyMesh
#   - Boundary data in constant/boundaryData
#   - Initial fields

# Initialize and run reconstruction (recommended: inlet-outlet BC)
```

```
cd ../ufr2-02-subset
spaceTimeWindowInitCase -sourceCase ../ufr2-02 -inletOutletBC

pimpleFoam

# Alternative: fixed outlet direction for steady-mean flows
# spaceTimeWindowInitCase -sourceCase ../ufr2-02 \
#     -outletDirection "(1 0 0)" \
#     -correctMassFlux
```

Listing 38: Running the example case

A.7 Validation

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

i Note

The reconstruction should exactly 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.

B Field Selection by Turbulence Model

When configuring the extraction, include all fields required by your turbulence model. Table 15 lists recommended fields for common models.

Table 15: 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)`

References

[1] Alin-Adrian Anton. *Space-Time Window Reconstruction in High-Performance Numeric Simulations: Application for CFD*. PhD thesis, Universitatea Politehnica Timişoara, Timişoara, Romania, November 2011. URL https://dspace.upt.ro/jspui/handle/123456789/643.

[2] ERCOFTAC. Classic collection database: Case 043 – flow around a square cylinder. http://cfd.mace.manchester.ac.uk/ercoftac/, 2024. Mesh generation script by Niklas Nordin.

[3] 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.