Software for Estuarine Circulation Modelling

The Estuarine Circulation Modeling software is a desktop software written to simulate and analyze estuarine hydrodynamics, stratification, turbulence, sediment transport, tidal dynamics, and wave-current interactions. It integrates numerical solvers, turbulence models, and visualization tools to model complex estuarine processes. The software supports both structured and unstructured grids, multiple coordinate systems (sigma and z-level), and advanced numerical schemes like Total Variation Diminishing (TVD).

Functionalities

1. **Core Estuarine Model**:
    - Manages parameters: estuary length (1000 m), depth (10 m), tidal amplitude (1 m), tidal period (12 hours), salinity (river: 0 PSU, ocean: 35 PSU), temperature (river: 25°C, ocean: 20°C).
    - Supports hydrostatic and non-hydrostatic modes with a Reynolds-Averaged Navier-Stokes (RANS) solver.
    - Tracks salt wedge position and provides methods to retrieve salinity, temperature, and velocity profiles.
2. **Hydrodynamic Solver**:
    - Solves 2D shallow water dynamics on a 100×100 grid for water level and velocities.
    - Incorporates tidal forcing, wind stress, Coriolis force, and bottom friction.
    - Applies river (x=0) and ocean (x=1000 m) boundary conditions.
    - Handles wet/dry dynamics and wave-current interactions.
3. **2D Shallow Water Equations**:
    - Implements a 2D shallow water model with UI visualization of water level, velocity, and salinity.
    - Allows control over tidal amplitude, period, wind speed, wind direction, wave height, and grid size.
    - Supports wet/dry transitions and wave-enhanced friction.
4. **Total Variation Diminishing**:
    - Uses TVD scheme with Harten-Lax-van Leer (HLL) flux to advect salinity, temperature, turbulent kinetic energy (k), and dissipation rate (ϵ) or specific dissipation rate (ω).
    - Supports structured (200×100) and unstructured (triangular mesh, 40×25 nodes) grids with bathymetry (shallower near river, $x < 200$ m).

- Provides UI visualization: plan view, cross-section, contour, and quiver plots.
- Tracks fields: salinity, temperature, velocity, density, k, ϵ/ω, eddy viscosity, and diffusivity.

5. **Stratification**:
   - Models 1D vertical stratification of density, salinity, temperature, and passive scalars on a 100-point grid.
   - Computes gradient Richardson number and adjusts eddy viscosity using k-ϵ, k-ω, or constant turbulence models.
   - UI controls mixing coefficient, critical Richardson number, and river scalar concentration.
6. **Baroclinic Flow**:
   - Simulates buoyancy-driven flows due to density gradients.
   - Updates velocity fields using finite differences, incorporating buoyancy effects.
7. **Passive Scalar Transport**:
   - Solves 1D advection-diffusion for passive scalars (e.g., pollutants).
   - Integrates with baroclinic flow for consistent transport.
8. **Comprehensive Forcing**:
   - Combines tidal, wind, and wave forcing.
   - UI adjusts tidal amplitude, wind speed, wave height, and visualizes velocity and water level fields.
9. **Wave-Current Interaction**:
   - Computes Stokes drift velocities using linear wave theory.
   - Calculates wave-enhanced bottom friction via a simplified Grant-Madsen model.
   - Supports dynamic updates of wave height, direction, period, and depth.
10. **Wind Forcing**:
    - Computes wind drag coefficient and stress components (τ_x, τ_y).
    - Parameterizes drag coefficient based on wind speed and wave height.
11. **Wet & Dry Algorithm**:
    - Manages wet/dry transitions for tidal flats using a minimum depth threshold (D_{min}).
    - Sets velocities, water level, and salinity to zero in dry cells.
    - Ensures mass conservation via flux divergence corrections.

12. **Simpson-Hunter Mechanism**:
    - Models tidal straining and internal tide effects in a sigma-coordinate system.
    - Computes Stokes drift, vertical velocity, and turbulent kinetic energy (TKE) production.
13. **Asymmetric Tidal Mixing**:
    - Simulates asymmetric tidal mixing effects on salinity and velocity fields.
    - Uses a `Cell` class to manage local properties (salinity, velocity, TKE).
14. **Bifurcated Estuary Model**:
    - Models circulation in bifurcated channels, accounting for branching flow dynamics.
15. **Equations of State**:
    - Computes water density based on salinity and temperature.
16. **Vertical Discretization (`VerticalDiscretization`)**:
    - Supports sigma (terrain-following) and z-level (fixed layers) coordinate systems.
    - Computes metric terms (z_ξ, z_η) for grid transformations.
17. **Richardson Number and SSI Mixing**:
    - Computes gradient Richardson number and strain-induced mixing for turbulence closure.
    - Supports k-ϵ and k-ω models.
18. **Large Eddy Simulation**:
    - Implements LES with a Smagorinsky subgrid model.
    - UI controls Smagorinsky coefficient and visualizes velocity/vorticity fields.
19. **Lattice Boltzmann LES**:
    - Uses Lattice Boltzmann Method (LBM) with a D3Q19 lattice for LES.
    - UI adjusts grid size and relaxation time, visualizing vorticity contours.
20. **Spectral Analyzer**:
    - Performs spectral analysis (Welch's method) and EOF/POD analysis on variables like Richardson number, velocity, and salinity.
    - UI controls variable selection, window type (Hanning, Hamming, Blackman, Rectangular), and visualization (PSD, heatmaps, modes).
21. **Multi-Fraction Sediment Transport**:
    - Models bedload and suspended load for multiple grain sizes.

- UI adjusts sediment properties (grain size, settling velocity) and visualizes concentration profiles.

22. **Adaptive Mesh Refinement**:
    - Refines grid based on velocity gradients or water level changes.
    - Dynamically adjusts resolution to capture fine-scale features.
23. **Visualization Renderer**:
    - Renders 2D visualizations of water level, salt wedge, salinity, temperature, passive scalar, and velocity profiles.

Simulation Logic

1. **Grid and Field Initialization**:
    - Structured grids (e.g., 200×100 in `TotalVariationDiminishing`, 100×100 in `HydrodynamicSolver`) or unstructured triangular meshes (40×25 nodes).
    - Sigma or z-level coordinates account for bathymetry (shallower near river, $x < 200$ m).
    - Fields (salinity, temperature, velocity, density, TKE, ϵ/ω) initialized with profiles (e.g., salinity: 0 PSU at river to 35 PSU at ocean).
2. **Forcing and Boundary Conditions**:
    - Tidal forcing via sinusoidal water level variations.
    - Wind forcing (`WindForcing`) and wave-current interactions (`WaveCurrentIntera` add stresses.
    - River ($x = 0$, low salinity/temperature) and ocean ($x = 1000$ m, high salinity/temperature) boundaries.
3. **Numerical Solvers**:
    - `HydrodynamicSolver` and `ShallowWaterEq2D`: Semi-implicit finite differences for shallow water dynamics ($\Delta t = 0.1$ s).
    - `TotalVariationDiminishing`: TVD scheme with HLL flux for stable advection.
    - `LargeEddySim`: Smagorinsky subgrid model for LES.
    - `LatticeBoltzmannLES`: LBM with D3Q19 lattice for turbulence.
    - `PassiveScalarTransportEq` and `Stratification`: Finite differences for advection-diffusion.
4. **Turbulence Modeling**:
    - k-ϵ and k-ω models compute TKE and dissipation/specific dissipation.

- Eddy viscosity modulated by gradient Richardson number.

5. **Transport and Stratification**:
    - Scalars (e.g., pollutants) advected/diffused using velocity fields from `BaroclinicFlow` or `HydrodynamicSolver`.
    - Density gradients drive baroclinic flows in `BaroclinicFlow`.
6. **Wet/Dry Dynamics**:
    - `WetAndDryAlgo` updates cell status based on depth threshold (D_{min}).
    - Adjusts fluxes to prevent flow into dry cells and ensures mass conservation.
7. **Visualization and Analysis**:
    - `VisualizationRenderer` plots plan views, cross-sections, contours, and quiver plots.
    - `SpectralAnalyzer` computes PSD and EOF/POD modes.
8. **Dynamic Updates**:
    - `AdaptiveMeshRef` refines grids based on gradients.
    - User inputs (e.g., tidal period, wind speed) updated via UI or programmatic methods.

Algorithms

1. **Initialization**:
    - Initialize grids: structured (200×100 or 100×100) or unstructured (40×25 nodes).
    - Set bathymetry: shallower near river ($x < 200$ m).
    - Initialize fields: salinity ($S(x) = 35 \cdot x/1000$), temperature ($T(x) = 25 - 5 \cdot x/1000$), velocity, TKE, ϵ/ω.
2. **Hydrodynamic Solver**:
    - Solve continuity: $\frac{\partial \eta}{\partial t} + \frac{\partial (Hu)}{\partial x} = 0$.
    - Solve momentum: $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} + \frac{1}{H}\frac{\partial (H\tau)}{\partial x} - \frac{C_f|u|u}{H} + F$.
    - Apply boundaries: river ($u = Q_r/(BH)$), ocean ($\eta = A\sin(\omega t)$).
    - Use semi-implicit finite differences, $\Delta t = 0.1$ s.
3. **TVD Advection**:

- Compute HLL flux: $F_{\text{HLL}} = \begin{cases} F_L, & s_L \geq 0 \\ F_R, & s_R \leq 0 \\ \frac{s_R F_L - s_L F_R + s_L s_R (U_R - U_L)}{s_R - s_L}, & \text{otherwise} \end{cases}$, where $s_L = \min(u_L - c_L, u_R - c_R, 0)$, $s_R = \max(u_L + c_L, u_R + c_R, 0)$.
- Apply Superbee limiter: $\psi(r) = \max(0, \min(2r, (1+r)/2, 2))$, $r = \frac{C_i - C_{i-1}}{C_{i+1} - C_i}$.
- Update salinity, temperature, k, ϵ/ω.

4. **Turbulence Modeling**:
   - k-ϵ model:
     - $\frac{\partial k}{\partial t} + u \cdot \nabla k = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k \right] + P_k - \epsilon$.
     - $\frac{\partial \epsilon}{\partial t} + u \cdot \nabla \epsilon = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\epsilon} \right) \nabla \epsilon \right] + C_{1\epsilon} \frac{\epsilon}{k} P_k - C_{2\epsilon} \frac{\epsilon^2}{k}$.
     - $\nu_t = C_\mu \frac{k^2}{\epsilon}$, $P_k = \nu_t \left(\frac{\partial u}{\partial x} \right)^2$.
   - k-ω model:
     - $\frac{\partial k}{\partial t} + u \cdot \nabla k = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k \right] + P_k - \beta^* k \omega$.
     - $\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\omega} \right) \nabla \omega \right] + \alpha_\omega \frac{\omega}{k} P_k - \beta \omega^2$.
     - $\nu_t = \frac{k}{\omega}$.
   - Richardson number: $Ri = \frac{N^2}{S^2}$, $N^2 = -\frac{g}{\rho_0} \frac{\partial \rho}{\partial z}$, $S^2 = \left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2$.
   - Adjust ν_t: $\nu_t = \frac{\nu_0}{1 + 10Ri}$ (k-ϵ) or $\nu_t = \nu_0 \frac{1 + 0.5 \min(Ri, Ri_{\text{crit}})}{1 + Ri}$ (k-ω).
5. **Wet/Dry Algorithm**:
   - Check depth: cell is wet if $H = \eta + h \geq D_{\text{min}}$, else dry.
   - Set dry cell values: $u = v = \eta = S = 0$.
   - Limit fluxes: $u_{i,j} = \min(u_{i,j}, 0)$ if eastern neighbor is dry, etc.
   - Update water level: $\Delta \eta = \frac{\text{Flux}_{\text{in}} - \text{Flux}_{\text{out}}}{\Delta x \Delta y}$.
6. **Spectral Analysis**:
   - Welch's method for PSD:
     - Window: Hanning ($w_i = 0.5(1 - \cos(2\pi i/(N-1)))$), Hamming, Blackman, or Rectangular.
     - Segment: $\text{segmentData}_i = \text{data}_{\text{start}+i} \cdot w_i$.
     - PSD: $\text{PSD}_i = \frac{1}{\text{windowPower} \cdot \text{samplingRate}} \cdot |\text{FFT}(\text{segmentData})_i|^2 / \text{numSegments}$.
     - Frequencies: $f_i = \frac{i}{\text{samplingRate} \cdot \text{segmentLength}}$.
   - EOF/POD via SVD:
     - Data matrix: $\text{dataMatrix}_{s,t} = \text{data}_t[s, 0] - \text{mean}_t$.

- Power iteration: iterate $u_i = \sum_j \text{dataMatrix}_{i,j} v_j$, normalize u; $v_j = \sum_i \text{dataMatrix}_{i,j} u_i$, normalize v.
- Singular value: $\sigma = \sqrt{\sum_i (\text{Av}_i)^2}$.
- Explained variance: $\text{variance}_m = \frac{\sigma_m^2}{\sum_i \sigma_i^2} \cdot 100$.

7. **Sediment Transport**:
   - Update concentrations: $\frac{\partial C_f}{\partial t} = -u\frac{\partial C_f}{\partial x} + D_f \frac{\partial^2 C_f}{\partial x^2} - w_{s,f} C_f$.
   - Compute bedload flux: $q_b = 8(\theta - \theta_c)^{1.5}\sqrt{\left(\frac{\rho_s}{\rho_0} - 1\right) g d^3}$ if $\theta > \theta_c$, else 0.
8. **Adaptive Mesh Refinement**:
   - Compute gradients: $G_{i,j} = \sqrt{\sum (\frac{\partial \phi}{\partial x})^2 + (\frac{\partial \phi}{\partial z})^2}$ for $\phi = S, T, u$.
   - Flag top 10% gradient cells.
   - Initialize 2×2 subgrids with bilinear interpolation.

Physical and Mathematical Models

Shallow Water Dynamics

- Continuity:

$$\frac{\partial \eta}{\partial t} + \frac{\partial (Hu)}{\partial x} = 0 \tag{1}$$

- Momentum:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} + \frac{1}{H}\frac{\partial (H\tau)}{\partial x} - \frac{C_f |u| u}{H} + F \tag{2}$$

where η is water surface elevation, $H = h + \eta$, h is bathymetry, u is velocity, $\tau = \nu_{\text{eff}} \frac{\partial u}{\partial x}$, $C_f = 0.002 - 0.005$, F includes Coriolis, wind, and baroclinic terms.

Baroclinic Flow

- Density:

$$\rho = \rho_0 [1 - \alpha (T - T_0) + \beta_S (S - S_0)] \tag{3}$$

where $\rho_0 = 1000\,\text{kg/m}^3$, $\alpha = 2 \times 10^{-4}\,/^\circ\text{C}$, $\beta_S = 8 \times 10^{-4}\,/\text{PSU}$, $T_0 = 20^\circ\text{C}$, $S_0 = 35\,\text{PSU}$.

- Baroclinic pressure gradient:

$$\frac{\partial p_b}{\partial x} = -g \int_{\eta}^{-h} \frac{\partial \rho}{\partial x}\, dz \tag{4}$$

Passive Scalar Transport

- Advection-diffusion:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} = \frac{\partial}{\partial x}\left(K\frac{\partial C}{\partial x}\right) \tag{5}$$

where $K = K_m + \frac{\nu_t}{\mathrm{Sc}_t}$, $K_m \approx 10^{-9}\,\mathrm{m}^2/\mathrm{s}$, $\mathrm{Sc}_t \approx 0.7$.

Turbulence Models

- k-ϵ model:

$$\frac{\partial k}{\partial t} + u \cdot \nabla k = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\nabla k\right] + P_k - \epsilon \tag{6}$$

$$\frac{\partial \epsilon}{\partial t} + u \cdot \nabla \epsilon = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\epsilon}\right)\nabla \epsilon\right] + C_{1\epsilon}\frac{\epsilon}{k}P_k - C_{2\epsilon}\frac{\epsilon^2}{k} \tag{7}$$

where $\nu_t = C_\mu \frac{k^2}{\epsilon}$, $P_k = \nu_t \left(\frac{\partial u}{\partial x}\right)^2$, $C_\mu = 0.09$, $\sigma_k = 1.0$, $\sigma_\epsilon = 1.3$, $C_{1\epsilon} = 1.44$, $C_{2\epsilon} = 1.92$.

- k-ω model:

$$\frac{\partial k}{\partial t} + u \cdot \nabla k = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\nabla k\right] + P_k - \beta^* k\omega \tag{8}$$

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\nabla \omega\right] + \alpha_\omega \frac{\omega}{k}P_k - \beta\omega^2 \tag{9}$$

where $\nu_t = \frac{k}{\omega}$.

Wave-Current Interaction

- Stokes drift:

$$u_s = \frac{a^2 \omega k \cosh(2kz)}{2\sinh^2(kd)} \tag{10}$$

where $L = \sqrt{gd}\cdot T$, $k = \frac{2\pi}{L}$, $\omega = \frac{2\pi}{T}$.

- Wave-enhanced friction:

$$C_d = C_{d0}(1 + \beta U_b), \quad U_b = \frac{a\omega}{\sinh(kd)} \tag{11}$$

where $\beta = 0.2$, $C_{d0} = 0.0025$, $C_d \leq 0.01$.

Wind Forcing

- Drag coefficient:

$$C_d = (0.75 + 0.067U_{10} + 0.1H_s) \times 10^{-3}, \quad 0.001 \leq C_d \leq 0.003 \tag{12}$$

- Wind stress:

$$\tau_x = \rho_{\text{air}} C_d U_{10}^2 \cos(\theta), \quad \tau_y = \rho_{\text{air}} C_d U_{10}^2 \sin(\theta) \tag{13}$$

where $\rho_{\text{air}} = 1.225\,\mathrm{kg/m}^3$.

Wet/Dry Algorithm

- Depth criterion: wet if $H = \eta + h \geq D_{\text{min}}$, else dry.
- Flux limiting: $u_{i,j} = \min(u_{i,j}, 0)$ if eastern neighbor is dry, etc.
- Mass conservation:

$$\Delta\eta = \frac{\text{Flux}_{\text{in}} - \text{Flux}_{\text{out}}}{\Delta x \Delta y}, \quad \text{Flux}_u = u(\eta + h)\Delta y \Delta t \tag{14}$$

Simpson-Hunter Mechanism

- Stokes drift:

$$u_s = a\omega e^{-2kz} \sin(\theta) \tag{15}$$

- Internal tide:

$$N^2 = -\frac{g}{\rho_0}\frac{\partial \rho}{\partial z}, \quad w_{\text{tide}} = A \sin(\theta)\sqrt{N^2} \cos\left(\frac{2\pi z}{H}\right) \tag{16}$$

- Tidal straining:

$$\frac{\partial S}{\partial t} = -Cu\frac{\partial S}{\partial x} \tag{17}$$

Asymmetric Tidal Mixing

- Navier-Stokes with Boussinesq:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{\nabla p}{\rho_0} + \nu \nabla^2 u + \frac{g(\rho - \rho_0)}{\rho_0} + f \times u - \frac{\tau_b}{\rho_0} \tag{18}$$

- Tidal asymmetry: flood factor 1.2, ebb factor 0.8.

Vertical Discretization

- Sigma coordinates: $z = \sigma H$, $\sigma \in [0, 1]$.
- Z-level: $\Delta z = H/(N_z - 1)$.
- Metric terms:

$$z_\xi \approx \frac{z_{i+1,j} - z_{i-1,j}}{2\Delta\xi}, \quad z_\eta \approx \frac{z_{i,j+1} - z_{i,j-1}}{2\Delta\eta} \tag{19}$$

Richardson Number and SSI Mixing

- Richardson number:

$$Ri = \frac{N^2}{S^2}, \quad N^2 = -\frac{g}{\rho_0}\frac{\partial \rho}{\partial z}, \quad S^2 = \left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2 \tag{20}$$

- Viscosity adjustment:

$$\nu_t = \frac{\nu_0}{1 + 10Ri} \quad \text{(k-}\epsilon\text{)}, \quad \nu_t = \nu_0 \frac{1 + 0.5 \min(Ri, 0.25)}{1 + Ri} \quad \text{(k-}\omega\text{)} \tag{21}$$

- TKE production:

$$P = \nu_t \left[\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right] \tag{22}$$

Lattice Boltzmann LES

- LBM:

$$f_i(x + c_i \Delta t, t + \Delta t) = f_i(x, t) + \frac{f_i^{\text{eq}}(x, t) - f_i(x, t)}{\tau} \tag{23}$$

- Smagorinsky:

$$\nu_t = (C_s \Delta)^2 \sqrt{2 S_{ij} S_{ij}}, \quad S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{24}$$

Spectral Analysis

- Synthetic data:
  - Richardson number: Value $= 0.5 + 0.3 \sin\left(\frac{2\pi t}{43200}\right) + 0.2 \sin\left(\frac{2\pi t}{17 \cdot 3600}\right) +$ noise.
  - Velocity: Value $= 0.1 + 0.05 \sin\left(\frac{2\pi t}{43200}\right) + 0.02 \sin\left(\frac{2\pi t}{21600}\right) +$ noise.
  - Salinity: $S_{x,y} = \left(30 + 2 \sin\left(\frac{2\pi t}{44712}\right)\right)(1 - 0.1(x + y)) +$ noise.
- Welch's PSD:

$$\text{PSD}_i = \frac{1}{\text{windowPower} \cdot \text{samplingRate}} \cdot \frac{|\text{FFT}(\text{segmentData})_i|^2}{\text{numSegments}} \tag{25}$$

- EOF/POD via SVD:

$$\text{dataMatrix}_{s,t} = \text{data}_t[s, 0] - \text{mean}_t \tag{26}$$

Sediment Transport

- Suspended load:

$$\frac{\partial C_f}{\partial t} = -u \frac{\partial C_f}{\partial x} + D_f \frac{\partial^2 C_f}{\partial x^2} - w_{s,f} C_f \tag{27}$$

- Bedload:

$$q_b = \begin{cases} 8 (\theta - \theta_c)^{1.5} \sqrt{\left(\frac{\rho_s}{\rho_0} - 1\right) g d^3}, & \theta > \theta_c \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

Estuarine Circulation Modeling Framework

Core Components and Initialization

The model represents an estuary with a length $L = 10,000\,\text{m}$ and depth $H = 10\,\text{m}$, discretized into $N = 100$ grid points with a spatial step $\Delta x = L/N$. The time step is $\Delta t = 10\,\text{s}$. Key parameters include:

- River inflow: $Q_r = 0.1\,\text{m}^3/\text{s}$
- Tidal amplitude: $A = 1.0\,\text{m}$
- Tidal period: $T = 43,200\,\text{s}$ (12 hours)
- Ocean salinity: $S_o = 35\,\text{PSU}$
- Ocean temperature: $T_o = 20\,^\circ\text{C}$

The model initializes salinity and temperature profiles (`salinityProfile`, `temperaturePro` across the grid. The salinity profile starts at zero (freshwater), and the temperature is set to the ocean temperature. A `HydrodynamicSolver` object handles velocity and eddy viscosity calculations, with options for a Reynolds-Averaged Navier-Stokes (RANS) solver or a simplified velocity model.

Functioning Logic

The `Update` method advances the simulation by one time step:

1. Updates the current time: $t \leftarrow t + \Delta t$.
2. Computes tidal velocity: $u_t = A \cos\left(\frac{2\pi t}{T}\right)$.
3. Computes water level: $\eta = A \sin\left(\frac{2\pi t}{T}\right)$.
4. Determines effective velocity, either via the RANS solver or a simplified model: $u_{\text{eff}} = u_t - \frac{Q_r}{H \cdot \Delta x}$.
5. Updates the salt wedge position: $x_s \leftarrow x_s + u_{\text{eff}} \Delta t$.
6. Updates salinity and temperature profiles using advection-diffusion equations.

Salt Wedge Dynamics

The salt wedge position x_s marks the boundary between freshwater and saline water, updated as:

$$x_s(t + \Delta t) = x_s(t) + u_{\text{eff}} \Delta t \tag{1}$$

The position is constrained within $[0, L]$.

Salinity and Temperature Profiles

The salinity $S(x,t)$ and temperature $T(x,t)$ profiles are updated using an advection-diffusion model:

$$\frac{\partial S}{\partial t} = -u_{\text{eff}} \frac{\partial S}{\partial x} + K \frac{\partial^2 S}{\partial x^2} \tag{2}$$

$$\frac{\partial T}{\partial t} = -u_{\text{eff}} \frac{\partial T}{\partial x} + K \frac{\partial^2 T}{\partial x^2} \tag{3}$$

where K is the diffusion coefficient (eddy viscosity from the RANS solver or a default $0.1\,\text{m}^2/\text{s}$). These are discretized using finite differences:

$$\text{Advection: } \frac{S_{i+1} - S_{i-1}}{2\Delta x} \tag{4}$$

$$\text{Diffusion: } K \frac{S_{i+1} - 2S_i + S_{i-1}}{\Delta x^2} \tag{5}$$

The profiles are constrained: $0 \leq S \leq S_o$, $0 \leq T \leq T_o$.

Hydrodynamic Solver

The `HydrodynamicSolver` computes velocity and eddy viscosity. When `UseRANSSolver` is enabled, it employs a RANS model, optionally with non-hydrostatic corrections. Otherwise, a simplified velocity is used. The solver provides velocity profiles and eddy viscosity at specific points, influencing mixing processes.

Key Outputs

The model provides:

- Salinity at a point: $S(x)$, interpolated from the profile.
- Temperature at a point: $T(x)$.
- Maximum salinity: $\max(S_i)$.
- Velocity and salinity/temperature profiles.
- Eddy viscosity and non-hydrostatic status.

Physical and Mathematical Models

The model simulates estuarine circulation driven by tidal forcing and river inflow. The tidal velocity and water level follow harmonic functions:

$$u_t = A \cos\left(\frac{2\pi t}{T}\right) \tag{6}$$

$$\eta = A \sin\left(\frac{2\pi t}{T}\right) \tag{7}$$

The salt wedge dynamics reflect the balance between river outflow and tidal inflow, while salinity and temperature evolve via advection-diffusion, capturing mixing processes. The RANS solver enhances realism by modeling turbulent mixing.

Hydrodynamic Solver

Core Components and Initialization

The `HydrodynamicSolver` class, implemented in C#, models the hydrodynamics of estuarine flow, incorporating momentum, pressure, and turbulence effects. The model discretizes an estuary of length $L = 10,000\,\text{m}$ and depth $H = 10\,\text{m}$ into $N = 100$ grid points, with spatial step $\Delta x = L/N$ and time step $\Delta t = 10\,\text{s}$. Key parameters include:

- Kinematic viscosity: $\nu = 10^{-6}\,\text{m}^2/\text{s}$
- Coriolis parameter: $f = 10^{-4}\,\text{s}^{-1}$
- Reference density: $\rho = 1000\,\text{kg}/\text{m}^3$
- Gravitational acceleration: $g = 9.81\,\text{m}/\text{s}^2$
- Resolution threshold: $\Delta x \leq 200\,\text{m}$ for non-hydrostatic mode

The solver initializes arrays for velocity (u), pressure (p), turbulent kinetic energy (k), dissipation rate (ϵ), and eddy viscosity (ν_t). Initial conditions set velocity to zero, pressure to hydrostatic ($p = \rho g H$), $k = 10^{-3}\,\text{m}^2/\text{s}^2$, $\epsilon = 10^{-6}\,\text{m}^2/\text{s}^3$, and $\nu_t = 0.01\,\text{m}^2/\text{s}$. A `BaroclinicFlow` object computes density-driven pressure gradients.

Functioning Logic

The `Solve` method updates the flow field each time step:

1. Determines non-hydrostatic mode: enabled if $\Delta x \leq 200\,\text{m}$ or overridden by `UseNonHydrostaticOverride`.
2. Computes baroclinic pressure gradient using salinity and temperature profiles.
3. Solves the momentum equation for velocity.
4. Updates turbulent quantities using the $k - \epsilon$ model.
5. Updates pressure, either non-hydrostatically or hydrostatically based on water level.

Momentum Equation

The momentum equation includes advection, pressure gradient, baroclinic, diffusion, and Coriolis terms:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - \frac{1}{\rho}\frac{\partial p}{\partial x} - \frac{1}{\rho}\frac{\partial p_b}{\partial x} + (\nu + \nu_t)\frac{\partial^2 u}{\partial x^2} - fu \tag{1}$$

where p_b is the baroclinic pressure. Discretized:

$$\text{Advection: } u_i \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{2}$$

$$\text{Pressure gradient: } -\frac{p_{i+1} - p_{i-1}}{2\rho\Delta x} \tag{3}$$

$$\text{Diffusion: } (\nu + \nu_{t,i}) \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \tag{4}$$

Boundary conditions set $u_0 = Q_r/H$ (river inflow) and $u_{N-1} = u_t$ (tidal velocity).

Turbulence Model

The $k - \epsilon$ model governs turbulence:

$$\frac{\partial k}{\partial t} = -u\frac{\partial k}{\partial x} + P - \epsilon + \frac{\partial}{\partial x}\left(\frac{\nu + \nu_t}{\sigma_k}\frac{\partial k}{\partial x}\right) \tag{5}$$

$$\frac{\partial \epsilon}{\partial t} = -u\frac{\partial \epsilon}{\partial x} + C_{1\epsilon}\frac{\epsilon}{k}P - C_{2\epsilon}\frac{\epsilon^2}{k} + \frac{\partial}{\partial x}\left(\frac{\nu + \nu_t}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial x}\right) \tag{6}$$

where $P = \nu_t \left(\frac{\partial u}{\partial x}\right)^2$ is shear production, and constants are $C_\mu = 0.09$, $\sigma_k = 1.0$, $\sigma_\epsilon = 1.3$, $C_{1\epsilon} = 1.44$, $C_{2\epsilon} = 1.92$. Eddy viscosity is:

$$\nu_t = C_\mu \frac{k^2}{\epsilon} \tag{7}$$

Boundary conditions set k and ϵ at the domain edges equal to adjacent interior points.

Pressure Update

Pressure is updated based on the mode:

- Non-hydrostatic:

$$p_i = p_i - \rho \frac{u_{i+1} - u_{i-1}}{2\Delta x}\frac{\Delta x^2}{\Delta t} \tag{8}$$

- Hydrostatic:

$$p_i = \rho g \left(H + \eta \frac{i\Delta x}{L}\right) \tag{9}$$

where η is the water level.

Key Outputs

The solver provides:

- Velocity at a point: $u(x)$, interpolated from the profile.
- Velocity profile: u_i.
- Eddy viscosity at a point: $\nu_t(x)$.
- Non-hydrostatic status.

Shallow Water Equations 2D Solver

Core Components and Initialization

The `ShallowWaterEq2D` class, implemented in C#, simulates two-dimensional estuarine circulation using the shallow water equations, incorporating tidal forcing, river discharge, wind stress, storm surges, and salinity transport. The estuary is defined with length $L = 10,000\,\text{m}$, width W, and depth $H = 10\,\text{m}$, discretized into a grid of $n_x \times n_y$ points with spatial steps $\Delta x = L/(n_x - 1)$ and $\Delta y = W/(n_y - 1)$. Key parameters include:

- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Freshwater density: $\rho_0 = 1000\,\text{kg/m}^3$
- Ocean water density: $\rho_{\text{ocean}} = 1025\,\text{kg/m}^3$
- Kinematic viscosity: $\nu = 10^{-6}\,\text{m}^2/\text{s}$
- Salinity diffusion coefficient: $\kappa = 10^{-4}\,\text{m}^2/\text{s}$
- Eddy viscosity: $\nu_t = 0.01\,\text{m}^2/\text{s}$
- Coriolis parameter: $f = 10^{-4}\,\text{s}^{-1}$
- Bottom friction coefficient: $C_f = 0.0025$
- Atmospheric pressure gradient: $\nabla P_{\text{atm}} = 0.0001\,\text{Pa/m}$
- Seasonal salinity amplitude: $S_a = 2.0\,\text{PSU}$

The model initializes 2D arrays for velocity components (U, V), water surface elevation (η), and salinity (S). Initial conditions set $U = Q_r/(WH)$, $V = 0$, $\eta = A\sin(2\pi x/L)$, and $S = S_g x$, where Q_r is river discharge, A is tidal amplitude, and S_g is the salinity gradient.

Functioning Logic

The `Update` method advances the simulation by a time step determined by the CFL condition:

$$\Delta t = C \min\left(\frac{\Delta x}{|u_{\text{max}}| + \sqrt{gH}}, \frac{\Delta y}{|v_{\text{max}}| + \sqrt{gH}}, \frac{\Delta x^2}{2(\nu + \nu_t)}, \frac{\Delta y^2}{2(\nu + \nu_t)}\right) \tag{1}$$

where $C = 0.4$ is the Courant number. The method:

1. Computes tidal forcing: $\eta_t = A\sin(2\pi t/T)$, $u_t = A(2\pi/T)\cos(2\pi t/T)$.
2. Applies wind stress (τ_x, τ_y) via `WindForcing`.
3. Includes storm surge: $\eta_s = A_s e^{-t/86400}$.
4. Accounts for tide-surge interaction and wave effects.
5. Updates interior points using the shallow water equations and salinity transport.

6. Applies wetting and drying if enabled via `WetAndDryAlgo`.
7. Checks for numerical stability.

Shallow Water Equations

The model solves the 2D shallow water equations for momentum and continuity:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -g\frac{\partial \eta}{\partial x} - \frac{g}{\rho_0}\frac{\partial \rho}{\partial x} + (\nu + \nu_t)\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + fv + \frac{\tau_x}{\rho_0 H} - C_f\frac{|u|u}{H} + \frac{\partial P_{\text{atm}}}{\partial x} + u_w \tag{2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -g\frac{\partial \eta}{\partial y} - \frac{g}{\rho_0}\frac{\partial \rho}{\partial y} + (\nu + \nu_t)\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - fu + \frac{\tau_y}{\rho_0 H} - C_f\frac{|v|v}{H} \tag{3}$$

$$\frac{\partial \eta}{\partial t} = -(H + \eta)\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \tag{4}$$

where $u_w = 0.5 k_w \sqrt{gH}$ is wave-induced velocity, $k_w = 2\pi/(L/10)$. Advection terms use a minmod flux limiter:

$$\phi = \max(0, \min(1, r)), \quad r = \frac{q - q_{\text{upwind}}}{q_{\text{downwind}} - q + 10^{-10}} \tag{5}$$

Salinity Transport

Salinity evolves via an advection-diffusion equation:

$$\frac{\partial S}{\partial t} + u\frac{\partial S}{\partial x} + v\frac{\partial S}{\partial y} = (\kappa + \nu_t)\left(\frac{\partial^2 S}{\partial x^2} + \frac{\partial^2 S}{\partial y^2}\right) + 0.1\left|\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right| S \tag{6}$$

Discretized with flux limiters for advection and central differences for diffusion, salinity is constrained: $0 \leq S \leq 35$ PSU.

Boundary Conditions

Boundary conditions are:

- At $x = 0$: $u = Q_r/(WH)$, $v = 0$, $\eta = 0$, $S = 0$.
- At $x = L$: $u = u_t$, $v = 0$, $\eta = \eta_t + \eta_s + 0.05AA_s \sin(2\pi t/T)$, $S = 35 + S_a \sin(2\pi t/(365 \cdot 86400))$.
- At $y = 0, W$: Reflective conditions for u, η, and S; $v = 0$.

Wetting and Drying

If enabled, the `WetAndDryAlgo` sets $u = v = 0$ and maintains η and S in dry cells (depth below `minDepth`), preventing unphysical flow.

Key Outputs

The model provides:

- Velocity fields: $U(x, y)$, $V(x, y)$.
- Water surface elevation: $\eta(x, y)$.
- Salinity field: $S(x, y)$.

Total Variation Diminishing

Core Components and Initialization

The `TotalVariationDiminishing` class, implemented in C#, simulates estuarine circulation in a two-dimensional (x-z) domain using a total variation diminishing (TVD) numerical scheme. It supports both structured and unstructured grids, with visualization capabilities for salinity, temperature, and velocity fields. The estuary has a length $L = 1000\,\text{m}$ and depth $H = 10\,\text{m}$, discretized into a structured grid of 200×100 points ($\Delta x = L/200$, $\Delta z = H/100$) or an unstructured triangular mesh. Key parameters include:

- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Reference density: $\rho_0 = 1000\,\text{kg/m}^3$
- Thermal expansion coefficient: $\alpha = 2 \times 10^{-4}\,°\text{C}^{-1}$
- Saline contraction coefficient: $\beta_s = 8 \times 10^{-4}\,\text{PSU}^{-1}$
- Reference temperature: $T_0 = 20\,°\text{C}$
- Reference salinity: $S_0 = 35\,\text{PSU}$
- Tidal amplitude: $A = 0.3\,\text{m}$
- Tidal period: $T = 60\,\text{s}$
- Horizontal diffusion coefficient: $\kappa_h = 0.01\,\text{m}^2/\text{s}$

The model initializes fields for salinity (S), temperature (T), density (ρ), velocity components (u, w), turbulent kinetic energy (k), dissipation rate (ϵ) or specific dissipation rate (ω), eddy viscosity (ν_t), and eddy diffusivity (κ_t). Initial conditions for the structured grid are:

$$S(x,z) = 17.5\left(1 + \tanh\left(\frac{x-500}{50}\right)\right)\left(1 - 0.5\frac{z}{H}\right) \tag{1}$$

$$T(x,z) = \max\left(15, 20 + 2\left(1 - \tanh\left(\frac{x-500}{50}\right)\right) - 2\frac{z}{H}\right) \tag{2}$$

$$\rho(x,z) = \rho_0\left(1 - \alpha(T - T_0) + \beta_s(S - S_0)\right) \tag{3}$$

with $u = A(1 - z/H)$, $w = -0.02(z/H)$, $k = 10^{-4}\,\text{m}^2/\text{s}^2$, $\epsilon = C_\mu k^{1.5}/0.1$, $\omega = k/(0.09 \cdot 0.1)$, $\nu_t = C_\mu k^2/\epsilon$, and $\kappa_t = \nu_t/0.7$. Unstructured grids use a similar setup with bathymetry.

Functioning Logic

The `UpdateSimulation` method advances the simulation by a fixed time step $\Delta t = 0.025\,\text{s}$, updating fields via:

1. Velocity updates using tidal and buoyancy forcing.
2. Advection-diffusion of salinity, temperature, and turbulence quantities using a TVD scheme with HLL flux limiter.

3. Turbulence modeling with either $k-\epsilon$ or $k-\omega$ models.
4. Visualization updates for plan view, cross-section, contour, or quiver plots.

The simulation supports user interaction via a GUI, allowing control of visualization modes, grid type, depth, x-position, tidal period, and turbulence model.

Velocity Update

The velocity field is updated with tidal and buoyancy effects:

$$u(x,z,t) = A\sin\left(\frac{2\pi t}{T}\right)\left(1-\frac{z}{H}\right) \tag{4}$$

$$\frac{\partial w}{\partial t} = -\frac{g}{\rho_0}\frac{\partial \rho}{\partial x} + \nu_t \frac{\partial^2 w}{\partial z^2} \tag{5}$$

For unstructured grids, bathymetry adjusts the depth: $H(x) = H(0.5+0.5\,\mathrm{min}(1, x/200))$. The vertical velocity w is constrained: $-0.5 \le w \le 0.5\,\mathrm{m/s}$.

TVD Advection-Diffusion

Salinity, temperature, and turbulence quantities are updated using a TVD scheme with HLL flux limiter:

$$\frac{\partial \phi}{\partial t} + \frac{\partial (u\phi)}{\partial x} + \frac{\partial (w\phi)}{\partial z} = \kappa_h \frac{\partial^2 \phi}{\partial x^2} + \kappa_t \frac{\partial^2 \phi}{\partial z^2} + S_\phi \tag{6}$$

where ϕ is S, T, k, ϵ, or ω, and S_ϕ is a source term (e.g., river/ocean salinity/temperature sources). The HLL flux is:

$$F_{\text{HLL}} = \frac{s_R v\phi_L - s_L v\phi_R + s_L s_R(\phi_R - \phi_L)}{s_R - s_L}, \quad s_L = v - |v|, \quad s_R = v + |v| \tag{7}$$

A limiter ensures stability:

$$\phi = \mathrm{max}(0, \mathrm{min}(2r, \mathrm{min}(1, (1+r)/2))), \quad r = \frac{\phi - \phi_{\text{upwind}}}{\phi_{\text{downwind}} - \phi + 10^{-10}} \tag{8}$$

Turbulence Models

The solver supports $k-\epsilon$ and $k-\omega$ models:

$$\frac{\partial k}{\partial t} + u\frac{\partial k}{\partial x} + w\frac{\partial k}{\partial z} = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial z}\right) + P - \epsilon \quad (\text{k-}\epsilon) \tag{9}$$

$$\frac{\partial \epsilon}{\partial t} + u\frac{\partial \epsilon}{\partial x} + w\frac{\partial \epsilon}{\partial z} = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial z}\right) + C_{1\epsilon}\frac{\epsilon}{k}P - C_{2\epsilon}\frac{\epsilon^2}{k} \tag{10}$$

$$\frac{\partial k}{\partial t} + u\frac{\partial k}{\partial x} + w\frac{\partial k}{\partial z} = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial z}\right) + P - \beta^* k\omega \quad (\text{k-}\omega) \tag{11}$$

$$\frac{\partial \omega}{\partial t} + u\frac{\partial \omega}{\partial x} + w\frac{\partial \omega}{\partial z} = \frac{\partial}{\partial z}\left(\frac{\nu_t}{\sigma_\omega}\frac{\partial \omega}{\partial z}\right) + \alpha_\omega\frac{\omega}{k}P - \beta\omega^2 \tag{12}$$

where $P = P_{\text{shear}} + P_{\text{buoy}}$, with $P_{\text{shear}} = \nu_t \left(\frac{\partial u}{\partial z}\right)^2$, $P_{\text{buoy}} = -g\kappa_t \frac{\partial \rho}{\partial z}/\rho_0$. Constants are $C_\mu = 0.09$, $\sigma_k = 1.0$, $\sigma_\epsilon = 1.3$, $C_{1\epsilon} = 1.44$, $C_{2\epsilon} = 1.92$, $\sigma_{k,\omega} = 0.5$, $\sigma_\omega = 0.5$, $\beta^* = 0.09$, $\beta = 0.075$, $\alpha_\omega = 0.52$. Eddy viscosity is:

$$\nu_t = \begin{cases} C_\mu \frac{k^2}{\epsilon} & (\text{k-}\epsilon) \\ \frac{k}{\omega} & (\text{k-}\omega) \end{cases} \tag{13}$$

Eddy diffusivity is $\kappa_t = \nu_t / 0.7$.

Visualization

The GUI provides four visualization modes:

- **Plan View**: Plots salinity, temperature, and velocity magnitude at a selected depth.
- **Cross-Section**: Plots profiles at a selected x-position.
- **Contour Plot**: Displays salinity and temperature as color maps.
- **Quiver Plot**: Shows velocity vectors.

For unstructured grids, fields are interpolated over triangles. The output textbox displays maximum salinity, temperature, and velocity.

Stratification

Core Components and Initialization

The `Stratification` class, implemented in C#, models vertical stratification in an estuary, focusing on density-driven flows and passive scalar transport. The estuary spans a length $L = 10,000\,\text{m}$ and depth H, discretized into n grid points with spatial step $\Delta x = L/n$. Key parameters include:

- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Reference density: $\rho_0 = 1000\,\text{kg/m}^3$
- Ocean salinity: $S_{\text{ocean}} = 35\,\text{PSU}$
- Ocean temperature: $T_{\text{ocean}} = 20\,°\text{C}$
- River scalar concentration: $C_{\text{river}} = 1.0\,\text{kg/m}^3$
- Ocean scalar concentration: $C_{\text{ocean}} = 0.0\,\text{kg/m}^3$
- Salt wedge position: $x_s = 5000\,\text{m}$
- Base mixing coefficient: $\kappa = 0.1\,\text{m}^2/\text{s}$
- Critical Richardson number: $Ri_c = 0.25$
- Time step: $\Delta t = 3600\,\text{s}$ (1 hour)

The model initializes arrays for salinity (S), temperature (T), density (ρ), passive scalar concentration (C), and gradient Richardson number (Ri). Initial profiles are linear:

$$S(x) = \frac{x}{L} S_{\text{ocean}} \tag{1}$$

$$T(x) = 15 + \frac{x}{L}(T_{\text{ocean}} - 15) \tag{2}$$

$$C(x) = C_{\text{river}} \left(1 - \frac{x}{L}\right) \tag{3}$$

$$\rho(x) = \text{CalculateDensity}(S(x), T(x)) \tag{4}$$

with $S \in [0, 35]$, $T \in [0, 20]$, $C \in [0, C_{\text{river}}]$, and $\rho \in [1000, 1030]\,\text{kg/m}^3$.

Functioning Logic

The `UpdateSimulation` method advances the simulation by Δt, performing:

1. Velocity profile computation: $u(x) = 0.1(1 - x/L)$, clamped to $[-1, 1]\,\text{m/s}$.
2. Stratification computation via `ComputeStratification`.
3. Salinity, temperature, and passive scalar updates using `BaroclinicFlow` and `PassiveScalarTransportEq`.
4. Visualization of density and passive scalar profiles.

A GUI allows users to adjust the turbulence model ($k - \epsilon$, $k - \omega$, or constant), mixing coefficient, critical Richardson number, and river scalar concentration.

Stratification Computation

The `ComputeStratification` method calculates:

- **Density**: $\rho = \text{CalculateDensity}(S, T)$, clamped to $[1000, 1030]\,\text{kg/m}^3$, computed via the `BaroclinicFlow` class.
- **Gradient Richardson Number**:

$$Ri = \frac{(g/\rho_0)(\partial\rho/\partial x)}{(\partial u/\partial x)^2} \tag{5}$$

  where $\partial\rho/\partial x = (\rho_{i+1} - \rho_{i-1})/(2\Delta x)$, $\partial u/\partial x = (u_{i+1} - u_{i-1})/(2\Delta x)$, clamped to $[-100, 100]$.
- **Eddy Viscosity**: Adjusted based on the turbulence model:
  - $k - \epsilon$:

$$\nu_t = \frac{\nu_t}{1 + \max(0, Ri/Ri_c)} \tag{6}$$

  - $k - \omega$:

$$\nu_t = \nu_t \frac{1 + 0.5 \min(Ri, Ri_c)}{1 + Ri} \tag{7}$$

  - Constant:

$$\nu_t = \kappa \tag{8}$$

  Eddy viscosity is clamped to $[0, 1]\,\text{m}^2/\text{s}$.

Transport Equations

Salinity, temperature, and passive scalar are updated using transport equations solved by `BaroclinicFlow` and `PassiveScalarTransportEq`:

$$\frac{\partial\phi}{\partial t} + u\frac{\partial\phi}{\partial x} = \frac{\partial}{\partial x}\left(\nu_t \frac{\partial\phi}{\partial x}\right) \tag{9}$$

where ϕ represents S, T, or C. Boundary conditions are:

- At $x = L$: $S = S_{\text{ocean}}$, $T = T_{\text{ocean}}$, $C = C_{\text{ocean}}$
- At $x = 0$: $C = C_{\text{river}}$

Visualization

The GUI displays density and passive scalar profiles in a panel, with:

- Density (ρ, blue): Scaled to $[1000, 1030]\,\text{kg/m}^3$.
- Passive scalar (C, red): Scaled to $[0, C_{\text{river}}]\,\text{kg/m}^3$.
- Axes labeled with position (km) and simulation time (hours).

Controls allow starting, pausing, resetting, and adjusting parameters, with updates every 100 ms.

Physical and Mathematical Models

The `Stratification` class simulates estuarine stratification driven by density gradients from salinity and temperature, coupled with passive scalar transport. The following models and equations are utilized:

- **Velocity Profile**: A simplified linear velocity profile drives advection:

$$u(x) = 0.1 \left(1 - \frac{x}{L}\right), \quad u \in [-1, 1]\,\text{m/s} \tag{10}$$

- **Density Calculation**: Density is computed via the `BaroclinicFlow` class (implementation not shown), constrained to:

$$\rho \in [1000, 1030]\,\text{kg/m}^3 \tag{11}$$

- **Gradient Richardson Number**: Assesses stability of stratified flow:

$$Ri = \frac{(g/\rho_0)(\partial\rho/\partial x)}{(\partial u/\partial x)^2}, \quad \frac{\partial\rho}{\partial x} = \frac{\rho_{i+1} - \rho_{i-1}}{2\Delta x}, \quad \frac{\partial u}{\partial x} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{12}$$

  with $Ri \in [-100, 100]$.

- **Eddy Viscosity Adjustment**: Modulates mixing based on turbulence model:
  - $k - \epsilon$:

$$\nu_t = \frac{\nu_t}{1 + \max(0, Ri/Ri_c)} \tag{13}$$

  - $k - \omega$:

$$\nu_t = \nu_t \frac{1 + 0.5 \min(Ri, Ri_c)}{1 + Ri} \tag{14}$$

  - Constant:

$$\nu_t = \kappa \tag{15}$$

  with $\nu_t \in [0, 1]\,\text{m}^2/\text{s}$.

- **Transport Equation**: Governs evolution of salinity (S), temperature (T), and passive scalar (C):

$$\frac{\partial\phi}{\partial t} + u\frac{\partial\phi}{\partial x} = \frac{\partial}{\partial x}\left(\nu_t \frac{\partial\phi}{\partial x}\right) \tag{16}$$

  solved numerically by `BaroclinicFlow` and `PassiveScalarTransportEq` classes, with boundary conditions:
  - At $x = L$: $S = 35$ PSU, $T = 20$ °C, $C = 0\,\text{kg/m}^3$
  - At $x = 0$: $C = C_{\text{river}}$

  and constraints $S \in [0, 35]$, $T \in [0, 20]$, $C \in [0, C_{\text{river}}]$.

- **Initial Conditions**: Linear profiles for initialization:

$$S(x) = \frac{x}{L} \cdot 35 \tag{17}$$

$$T(x) = 15 + \frac{x}{L} \cdot 5 \tag{18}$$

$$C(x) = C_{\text{river}} \left(1 - \frac{x}{L}\right) \tag{19}$$

These models capture the interplay of advection, diffusion, and stratification, with turbulence modulated by the Richardson number. The simulation ensures numerical stability through clamping and provides real-time visualization of density and scalar profiles.

Baroclinic Flow

Core Components and Initialization

The `BaroclinicFlow` class, implemented in C#, models density-driven flows in an estuary, focusing on salinity and temperature transport influenced by baroclinic pressure gradients. The estuary spans a length L and depth H, discretized into n grid points with spatial step $\Delta x = L/n$. Key parameters include:

- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Reference density: $\rho_0 = 1000\,\text{kg/m}^3$
- Base horizontal diffusivity: $K_x = 0.1\,\text{m}^2/\text{s}$
- Base vertical diffusivity: $K_z = 0.01\,\text{m}^2/\text{s}$
- Mixing rate: $\gamma = 0.005\,\text{s}^{-1}$
- Turbulent length scale: $l = 10\,\text{m}$
- Critical Richardson number: $Ri_c = 0.25$
- Specific heat capacity of water: $C_p = 4184\,\text{J/kg}\cdot\text{K}$
- Stefan-Boltzmann constant: $\sigma = 5.67 \times 10^{-8}\,\text{W/m}^2\cdot\text{K}^4$
- Surface albedo: $\alpha = 0.06$
- Emissivity: $\epsilon = 0.97$
- Latent heat of vaporization: $L_v = 2.45 \times 10^6\,\text{J/kg}$
- Bowen ratio: $B = 0.61$
- Wind speed: $U = 5\,\text{m/s}$
- Cloud cover fraction: $C_c = 0.5$
- Atmospheric temperature: $T_a = 15\,°\text{C}$

The class initializes arrays for salinity, temperature, density, and baroclinic pressure gradients, with computations driven by the equation of state for density.

Functioning Logic

The `BaroclinicFlow` class computes:

1. Density using the equation of state via `EqOfState.ComputeDensity`.
2. Baroclinic pressure gradient based on density gradients.
3. Eddy diffusivities (K_x, K_z) modulated by velocity shear and stratification.
4. Surface heat flux incorporating shortwave, longwave, sensible, and latent heat.
5. Salinity and temperature transport using a semi-implicit scheme with Van Leer flux limiter.

The model supports external eddy viscosity inputs for compatibility with turbulence models.

Baroclinic Pressure Gradient

The `ComputeBaroclinicGradient` method calculates the baroclinic pressure gradient:

$$\frac{\partial p_b}{\partial x} \approx g\rho h \frac{\partial \rho / \partial x}{\rho_0} \tag{1}$$

where:

- $\rho = \text{ComputeDensity}(S, T, P)$, with pressure $P = \rho_0 g h / 10^4$ dbar
- $\partial \rho / \partial x = (\rho_{i+1} - \rho_{i-1})/(2\Delta x)$

Boundary conditions set $\partial p_b / \partial x$ at endpoints equal to adjacent interior points.

Eddy Diffusivities

The `ComputeEddyDiffusivities` method computes:

- Horizontal diffusivity:

$$K_x = \nu_t + l\sqrt{\left|\frac{\partial u}{\partial x}\right|}, \quad \frac{\partial u}{\partial x} = \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{2}$$

- Vertical diffusivity:

$$K_z = \frac{\nu_t + K_z^{\text{base}}}{1 + \max(0, Ri/Ri_c)}, \quad Ri = \frac{g(\partial \rho/\partial x)/\rho_0}{(\partial u/\partial x)^2} \tag{3}$$

where ν_t is the eddy viscosity (default $K_x^{\text{base}} = 0.1\,\text{m}^2/\text{s}$).

Surface Heat Flux

The `ComputeSurfaceHeatFlux` method calculates the total heat flux:

$$Q_{\text{surface}} = Q_{\text{sw}} - Q_{\text{lw}} - Q_{\text{sensible}} - Q_{\text{latent}} \tag{4}$$

where:

- Shortwave: $Q_{\text{sw}} = Q_{\text{solar}}(1 - \alpha)(1 - C_c)\max(0, \cos(2\pi t / T_d))$, $T_d = 24 \cdot 3600\,\text{s}$
- Longwave: $Q_{\text{lw}} = \epsilon\sigma(T_w^4 - T_a^4)$, with temperatures in Kelvin
- Sensible: $Q_{\text{sensible}} = \rho_{\text{air}} C_{p,\text{air}} C_h U (T_w - T_a) B$
- Latent: $Q_{\text{latent}} = \rho_{\text{air}} L_v C_e U (q_s - q_a)/100$, with q_s, q_a from Tetens formula

The flux is scaled by velocity gradient and converted to a temperature source term:

$$Q_t = \frac{Q_{\text{surface}}(1 + |\partial u/\partial x|)}{\rho_0 C_p h} \tag{5}$$

Transport Equations

Salinity and temperature are updated using:

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (u\phi) = \nabla \cdot (K_x \frac{\partial \phi}{\partial x} + K_z \frac{\partial \phi}{\partial z}) + Q_\phi \tag{6}$$

where ϕ is S or T, and Q_ϕ includes source/sink terms:

- Salinity: $Q_s = -\gamma |\partial u/\partial x| S$ (river, $x < x_s$) or $\gamma |\partial u/\partial x| (S_{\text{ocean}} - S)$ (ocean, $x \geq x_s$)
- Temperature: Q_t from surface heat flux, plus vertical mixing $\alpha_z (T_{\text{ocean}} - T)$

Advection uses the Van Leer limiter:

$$\phi_{\text{interface}} = \phi_i + 0.5 \phi_{\text{VL}} (\phi_{i+1} - \phi_i), \quad \phi_{\text{VL}} = \max(0, \min(2r/(1+r), 2)), \quad r = \frac{\phi_i - \phi_{i-1}}{\phi_{i+1} - \phi_i + 10^{-10}} \tag{7}$$

Diffusion is solved semi-implicitly (Crank-Nicolson) using a tridiagonal system:

$$a_i = -\alpha_x, \quad b_i = 1 + 2\alpha_x, \quad c_i = -\alpha_x, \quad \alpha_x = \frac{\kappa_x \Delta t}{2\Delta x^2} \tag{8}$$

$$d_i = \phi_i + \Delta t \left(-\frac{f_{i+1} - f_i}{\Delta x} + Q_\phi + \alpha_z (\phi_{\text{boundary}} - \phi_i) \right) + \alpha_x (\phi_{i+1} - 2\phi_i + \phi_{i-1}) \tag{9}$$

Boundary conditions: $S = 0$, $T = 15\,°\text{C}$ at $x = 0$; $S = S_{\text{ocean}}$, $T = T_{\text{ocean}}$ at $x = L$.

Physical and Mathematical Models

The `BaroclinicFlow` class simulates density-driven estuarine flows using the following models:

- **Density Calculation**: Equation of state:

$$\rho = \text{ComputeDensity}(S, T, P), \quad P = \frac{\rho_0 g h}{10^4}\ \text{dbar} \tag{10}$$

- **Baroclinic Pressure Gradient**:

$$\frac{\partial p_b}{\partial x} = g \rho h \frac{\rho_{i+1} - \rho_{i-1}}{2\rho_0 \Delta x} \tag{11}$$

- **Eddy Diffusivities**:

$$K_x = \nu_t + l \sqrt{\left| \frac{u_{i+1} - u_{i-1}}{2\Delta x} \right|} \tag{12}$$

$$K_z = \frac{\nu_t + 0.01}{1 + \max(0, Ri/0.25)}, \quad Ri = \frac{g(\rho_{i+1} - \rho_{i-1})/(2\rho_0 \Delta x)}{(u_{i+1} - u_{i-1})^2/(2\Delta x)^2} \tag{13}$$

- **Surface Heat Flux**:

$$Q_{\text{sw}} = 1000(1 - 0.06)(1 - 0.5)\,\max(0, \cos(2\pi t/86400)) \tag{14}$$

$$Q_{\text{lw}} = 0.97 \cdot 5.67 \times 10^{-8}(T_w^4 - T_a^4) \tag{15}$$

$$Q_{\text{sensible}} = 1.225 \cdot 1005 \cdot 0.0012 \cdot 5(T_w - 15) \cdot 0.61 \tag{16}$$

$$Q_{\text{latent}} = 1.225 \cdot 2.45 \times 10^6 \cdot 0.0015 \cdot 5(q_s - q_a)/100 \tag{17}$$

$$Q_t = \frac{Q_{\text{sw}} - Q_{\text{lw}} - Q_{\text{sensible}} - Q_{\text{latent}}(1 + |\partial u/\partial x|)}{\rho_0 C_p h} \tag{18}$$

  where $q_s = 6.1078 \cdot 10^{7.5T_w/(T_w+237.3)}$, $q_a = 0.8 \cdot 6.1078 \cdot 10^{7.5T_a/(T_a+237.3)}$.

- **Transport Equations**: For salinity and temperature:

$$\frac{\partial \phi}{\partial t} + \frac{\partial (u\phi)}{\partial x} = \frac{\partial}{\partial x}\left(K_x \frac{\partial \phi}{\partial x}\right) + \frac{K_z(\phi_{\text{boundary}} - \phi)}{h^2} + Q_\phi \tag{19}$$

  with $Q_s = \gamma|\partial u/\partial x|(-S$ or $S_{\text{ocean}} - S)$, Q_t from heat flux, and Van Leer flux limiter for advection.

- **Tridiagonal Solver**: Thomas algorithm for semi-implicit diffusion:

$$c'_i = \frac{c_i}{b_i - a_i c'_{i-1}}, \quad d'_i = \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} \tag{20}$$

$$\phi_i = d'_i - c'_i \phi_{i+1} \tag{21}$$

These models capture baroclinic dynamics, heat exchange, and turbulent mixing, ensuring numerical stability through clamping and flux limiters.

Passive Scalar Transport Equation

Core Components and Initialization

The `PassiveScalarTransportEq` class, implemented in C#, models the transport of a passive scalar (e.g., a tracer or pollutant) in an estuary. The domain spans a length L, discretized into n grid points with spatial step $\Delta x = L/n$. Key parameters include:

- Estuary length: L
- Grid points: n
- Spatial step: $\Delta x = L/n$
- River boundary concentration: C_{river}
- Ocean boundary concentration: C_{ocean}
- Salt wedge position: x_s

The class initializes arrays for the passive scalar concentration (C) and relies on external inputs for velocity (u) and eddy diffusivity (κ).

Functioning Logic

The `SolvePassiveScalarTransport` method advances the passive scalar concentration over a time step Δt, performing:

1. Advection-diffusion computation using a finite difference scheme.
2. Application of boundary conditions at the river ($x = 0$) and ocean ($x = L$).
3. Mixing adjustment near the salt wedge position using a Gaussian weighting function.

Transport Computation

The method solves the advection-diffusion equation for the passive scalar concentration:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} = \frac{\partial}{\partial x}\left(\kappa\frac{\partial C}{\partial x}\right) \tag{1}$$

using an explicit finite difference scheme:

- **Advection term**: Central difference approximation:

$$u\frac{\partial C}{\partial x} \approx u_i\frac{C_{i+1} - C_{i-1}}{2\Delta x} \tag{2}$$

- **Diffusion term**: Second-order finite difference:

$$\frac{\partial}{\partial x}\left(\kappa\frac{\partial C}{\partial x}\right) \approx \frac{\kappa_{i+1}(C_{i+1} - C_i)/\Delta x - \kappa_i(C_i - C_{i-1})/\Delta x}{\Delta x} \tag{3}$$

- **Time update**: Explicit Euler step:

$$C_i^{n+1} = C_i^n + \Delta t \left(-u_i \frac{C_{i+1} - C_{i-1}}{2\Delta x} + \frac{\kappa_{i+1}(C_{i+1} - C_i) - \kappa_i(C_i - C_{i-1})}{\Delta x^2} \right) \tag{4}$$

Boundary conditions are:

- River ($x = 0$): $C_0 = C_{\text{river}}$
- Ocean ($x = L$): $C_{n-1} = C_{\text{ocean}}$

Mixing Near Salt Wedge

A Gaussian mixing factor is applied near the salt wedge position x_s:

$$m_i = \exp\left(-\frac{(i\Delta x - x_s)^2}{0.1L^2} \right) \tag{5}$$

The concentration is adjusted as:

$$C_i^{n+1} = (1 - m_i) C_i^{n+1} + m_i C_i^n \tag{6}$$

This smooths the concentration field near x_s, simulating enhanced mixing at the salt wedge interface.

Physical and Mathematical Models

The `PassiveScalarTransportEq` class simulates the transport of a passive scalar using the following models:

- **Advection-Diffusion Equation**:

$$\frac{\partial C}{\partial t} + u \frac{\partial C}{\partial x} = \frac{\partial}{\partial x} \left(\kappa \frac{\partial C}{\partial x} \right) \tag{7}$$

  where C is the scalar concentration, u is the velocity, and κ is the eddy diffusivity.

- **Finite Difference Discretization**:
  - Advection: $u_i \frac{C_{i+1} - C_{i-1}}{2\Delta x}$
  - Diffusion: $\frac{\kappa_{i+1}(C_{i+1} - C_i) - \kappa_i(C_i - C_{i-1})}{\Delta x^2}$
  - Time stepping: $C_i^{n+1} = C_i^n + \Delta t \left(-\text{advection} + \text{diffusion} \right)$
- **Boundary Conditions**:

$$C_0 = C_{\text{river}} \tag{8}$$

$$C_{n-1} = C_{\text{ocean}} \tag{9}$$

- **Salt Wedge Mixing**:

$$m_i = \exp\left(-\frac{(i\Delta x - x_s)^2}{0.1L^2}\right) \tag{10}$$

$$C_i^{n+1} = (1 - m_i)C_i^{n+1} + m_i C_i^n \tag{11}$$

These models capture the transport and mixing of a passive scalar in an estuarine environment, with numerical stability ensured by explicit time stepping and boundary constraints.

Comprehensive Forcing Mechanism

Core Components and Initialization

The `CompForcingMechanism` class, implemented in C#, provides a graphical interface for simulating estuarine circulation using three solver types: 1D Simplified, 2D Shallow Water, and 3D Navier-Stokes. The simulation domain is an estuary with length $L = 10000\,\text{m}$, width $W = 2000\,\text{m}$, and depth $H = 10\,\text{m}$, discretized into grids of $n_x = 50$, $n_y = 20$, and $n_z = 10$ for 3D, or $n = 100$ for 1D. Key parameters include:

- River discharge: $Q_r = 0.1\,\text{m}^3/\text{s}$
- Tidal amplitude: $A_t = 1.0\,\text{m}$, period: $T_t = 43200\,\text{s}$
- Wind speed: $U_w = 5.0\,\text{m/s}$, direction: $\theta_w = 0^\circ$
- Salinity gradient: $\partial S/\partial x = 0.0035\,\text{PSU/m}$
- Wave height: $H_w = 0.5\,\text{m}$, period: $T_w = 10\,\text{s}$
- Storm surge amplitude: $A_s = 0.0\,\text{m}$
- Minimum depth for wetting/drying: $h_{\text{min}} = 0.01\,\text{m}$
- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Freshwater density: $\rho_0 = 1000\,\text{kg/m}^3$, ocean density: $\rho_{\text{ocean}} = 1025\,\text{kg/m}^3$
- Kinematic viscosity: $\nu = 10^{-6}\,\text{m}^2/\text{s}$
- Salinity diffusion coefficient: $\kappa = 10^{-4}\,\text{m}^2/\text{s}$
- Eddy viscosity: $\nu_e = 0.01\,\text{m}^2/\text{s}$
- Coriolis parameter: $f = 10^{-4}\,\text{s}^{-1}$
- Friction coefficient: $C_f = 0.0025$
- Atmospheric pressure gradient: $\partial P_a/\partial x = 0.0001\,\text{Pa/m}$
- Seasonal salinity amplitude: $S_a = 2.0\,\text{PSU}$

The class initializes a Windows Forms interface with input controls for parameters, a visualization panel, and an output console. It supports dynamic switching between solvers and optional wetting/drying algorithms.

Functioning Logic

The `CompForcingMechanism` class manages:

1. **User Interface**: Allows input of physical parameters and solver selection (1D, 2D, or 3D).
2. **Simulation Initialization**: Sets up grids and initial conditions based on solver type.

3. **Time Stepping**: Uses a Courant-Friedrichs-Lewy (CFL) condition with $C = 0.4$ to compute adaptive time steps Δt.
4. **Forcing Mechanisms**: Incorporates river discharge, tides, wind stress, waves, storm surge, baroclinic effects, Coriolis force, and friction.
5. **Numerical Solvers**: Updates velocity, salinity, water level, and temperature using explicit schemes with flux limiters.
6. **Visualization**: Plots velocity (blue), salinity (red), and water level (green) profiles.
7. **Stability Checks**: Detects numerical instabilities (NaN or Infinity) and stops the simulation if detected.

Solver Initialization

The `InitializeSimulation` method configures the solver based on the selected type:

- **3D Navier-Stokes**: Initializes 3D arrays for velocity components (u, v, w), pressure (p), salinity (S), and temperature (T). Spatial steps are $\Delta x = L/(n_x - 1)$, $\Delta y = W/(n_y - 1)$, $\Delta z = H/(n_z - 1)$. Initial conditions:

$$u_{i,j,k} = Q_r/(WH), \quad v_{i,j,k} = w_{i,j,k} = 0 \tag{1}$$

$$p_{i,j,k} = \rho_0 g \eta (1 - k\Delta z/H), \quad \eta = A_t \sin(2\pi x/L) + A_s e^{-x/L} \tag{2}$$

$$S_{i,j,k} = (\partial S/\partial x)x, \quad T_{i,j,k} = 20\,\text{°C} \tag{3}$$

- **2D Shallow Water**: Instantiates `ShallowWaterEq2D` with bathymetry $h_{i,j} = H(1 - x/L)$ if wetting/drying is enabled, using `WetAndDryAlgo`.
- **1D Simplified**: Initializes arrays for velocity (u), salinity (S), and water level (η) with $\Delta x = L/(n-1)$:

$$u_i = Q_r/H, \quad S_i = (\partial S/\partial x) i \Delta x, \quad \eta_i = A_t \sin(2\pi x/L) \tag{4}$$

CFL Time Step

The `ComputeCFLTimeStep` method ensures numerical stability:

- **3D**: $\Delta t = C \min\left(\frac{\Delta x}{|u|_{\max}}, \frac{\Delta y}{|v|_{\max}}, \frac{\Delta z}{|w|_{\max}}, \frac{\Delta x^2}{2(\nu+\nu_e)}, \frac{\Delta y^2}{2(\nu+\nu_e)}, \frac{\Delta z^2}{2(\nu+\nu_e)}\right)$
- **2D**: Delegates to `ShallowWaterEq2D.ComputeCFLTimeStep`.
- **1D**: $\Delta t = C \min\left(\frac{\Delta x}{|u|_{\max}}, \frac{\Delta x^2}{2(\nu+\nu_e)}\right)$

The time step is constrained to $\Delta t \geq 0.01$ s.

3D Navier-Stokes Solver

The `UpdateSimulation` method for 3D solves:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho_0}\frac{\partial p}{\partial x} + (\nu + \nu_e)\nabla^2 u + fv - \frac{g}{\rho_0}\frac{\partial \rho}{\partial x} + F_u \tag{5}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} = -\frac{1}{\rho_0}\frac{\partial p}{\partial y} + (\nu + \nu_e)\nabla^2 v - fu + F_v \tag{6}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = -\frac{1}{\rho_0}\frac{\partial p}{\partial z} + (\nu + \nu_e)\nabla^2 w - g \tag{7}$$

$$\frac{\partial S}{\partial t} + u\frac{\partial S}{\partial x} + v\frac{\partial S}{\partial y} + w\frac{\partial S}{\partial z} = \kappa\nabla^2 S + 0.1\left|\frac{\partial u}{\partial x}\right| S \tag{8}$$

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} + w\frac{\partial T}{\partial z} = \kappa\nabla^2 T \tag{9}$$

where $\rho = \rho_0 + S(\rho_{\text{ocean}} - \rho_0)/35 - 0.2(T - 20)$, and forcing terms include:

- Wind stress: $\tau_x = 0.001\rho_0 U_w^2 \cos(\theta_w \pi/180)$, $\tau_y = 0.001\rho_0 U_w^2 \sin(\theta_w \pi/180)$
- Wave velocity: $u_w = 0.5 H_w k\sqrt{gH}$, $k = 2\pi/(L/10)$
- Quadratic drag: $-C_f|u|u/H$
- Atmospheric pressure gradient: $\partial P_a/\partial x/\rho_0$

Advection uses upwind schemes with a minmod flux limiter:

$$\phi_{\text{lim}} = \max(0, \min(1, r)), \quad r = \frac{\phi - \phi_{\text{upwind}}}{\phi_{\text{downwind}} - \phi + 10^{-10}} \tag{10}$$

Pressure is corrected iteratively to enforce incompressibility:

$$\nabla^2 p = \nabla\cdot\mathbf{u}, \quad p_{i,j,k} = \frac{p_{i+1,j,k} + p_{i-1,j,k} + p_{i,j+1,k} + p_{i,j-1,k} + p_{i,j,k+1} + p_{i,j,k-1} - \Delta x^2(\nabla \cdot \mathbf{u})}{6} \tag{11}$$

Boundary conditions:

$$x = 0: \quad u = Q_r/(WH), \quad v = w = 0, \quad S = 0, \quad T = 20 \tag{12}$$

$$x = L: \quad u = A_t(2\pi/T_t)\cos(2\pi t/T_t), \quad v = w = 0, \quad p = \rho_0 g(A_s e^{-t/86400} + 0.05 A_t A_s \sin(2\pi t/T_t)), \quad S \tag{13}$$

2D Shallow Water Solver

The `ShallowWaterEq2D` class updates velocity (u, v), water level (η), and salinity (S) using shallow water equations, with optional wetting/drying via `WetAndDryAlgo`. Forcing includes wind, tides, waves, and storm surge.

1D Simplified Solver

The 1D solver updates velocity (u), water level (η), and salinity (S):

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} - \frac{g}{\rho_0}\frac{\partial \rho}{\partial x} + \frac{\tau_x}{\rho_0 H} + u_w + 0.05A_t A_s \sin(2\pi t/T_t) - C_f|u|u/H + fu + \frac{\partial P_a/\partial x}{\rho_0} \tag{14}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}[(H+\eta)u] = A_t \sin(2\pi(t/T_t - x/\sqrt{gH})) - \epsilon\eta \tag{15}$$

$$\frac{\partial S}{\partial t} + u\frac{\partial S}{\partial x} = (\kappa + \nu_e)\frac{\partial^2 S}{\partial x^2} + 0.1\left|\frac{\partial u}{\partial x}\right| S \tag{16}$$

Boundary conditions:

$$x = 0: \quad u = Q_r/H, \quad S = 0 \tag{17}$$

$$x = L: \quad u = A_t(2\pi/T_t)\cos(2\pi t/T_t), \quad \eta = A_t \sin(2\pi t/T_t) + A_s e^{-t/86400}, \quad S = 35 + S_a \sin(2\pi t/(365 \cdot \tag{18}$$

Advection uses upwind schemes with minmod flux limiter, and salinity is smoothed near boundaries.

Visualization

The `visualizationPanel_Paint` method plots:

- Velocity (blue), salinity (red), and water level (green) as 1D profiles along the estuary length.
- For 3D, variables are averaged over y and z.
- For 2D, wet/dry cells are shown as light blue (wet) or tan (dry) backgrounds.

Values are scaled to the panel dimensions, with checks for NaN or Infinity.

Physical and Mathematical Models

- **3D Navier-Stokes**:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\frac{1}{\rho_0}\nabla p + (\nu + \nu_e)\nabla^2\mathbf{u} + \mathbf{f}\times\mathbf{u} - \frac{g}{\rho_0}\nabla\rho + \mathbf{F} \tag{19}$$

$$\nabla\cdot\mathbf{u} = 0, \quad \frac{\partial S}{\partial t} + \mathbf{u}\cdot\nabla S = \kappa\nabla^2 S + 0.1\left|\frac{\partial u}{\partial x}\right| S, \quad \frac{\partial T}{\partial t} + \mathbf{u}\cdot\nabla T = \kappa\nabla^2 T \tag{20}$$

- **2D Shallow Water**: Handled by `ShallowWaterEq2D` with wetting/drying.
- **1D Simplified**:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial \eta}{\partial x} - \frac{g}{\rho_0}\frac{\partial \rho}{\partial x} + F_{\text{total}}, \quad \frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}[(H+\eta)u] = F_\eta, \quad \frac{\partial S}{\partial t} + u\frac{\partial S}{\partial x} = (\kappa + \nu_e)\frac{\partial^2 S}{\partial x^2} + \tag{21}$$

- **Forcing Terms**: Wind stress, wave velocity, tide-surge interaction, quadratic drag, Coriolis, atmospheric pressure gradient.
- **Numerical Methods**: Explicit time stepping, upwind advection with min-mod limiter, iterative pressure correction (3D), and boundary smoothing.

The model integrates multiple physical processes, ensuring robust simulation of estuarine dynamics with user-configurable parameters and real-time visualization.

Wave Current Interaction

Core Components and Initialization

The `WaveCurrentInteraction` class, implemented in C#, models the interaction between waves and currents in an estuarine environment using linear wave theory and simplified wave-enhanced friction. The class is initialized with the following parameters:

- Wave height: H_w (m)
- Wave direction: θ_w (degrees, aligned with wind)
- Wave period: T_w (s)
- Water depth: h (m)
- Gravitational acceleration: $g = 9.81\,\mathrm{m/s^2}$

The constructor sets these parameters, and the `UpdateParameters` method allows dynamic updates during simulation.

Functioning Logic

The class provides two primary methods to account for wave-current interactions:

1. `ComputeStokesDrift`: Calculates Stokes drift velocities (u_s, v_s) based on linear wave theory, representing the net mass transport induced by waves.
2. `ComputeWaveEnhancedFriction`: Computes a wave-enhanced bottom friction coefficient using a simplified Grant-Madsen approach, accounting for increased turbulence due to wave orbital velocities.

Stokes Drift Computation

The `ComputeStokesDrift` method calculates Stokes drift velocities using linear wave theory for shallow water:

- **Wave characteristics**:
  - Approximate wavelength: $\lambda = \sqrt{gh}T_w$
  - Wave number: $k = 2\pi/\lambda$
  - Angular frequency: $\omega = 2\pi/T_w$
  - Wave amplitude: $a = H_w/2$
- **Stokes drift magnitude**: Near the surface ($z \approx 0$):

$$u_s = \frac{a^2 \omega k \cosh(2kz)}{2\sinh^2(kh)} \approx \frac{a^2 \omega k}{2\sinh^2(kh)} \tag{1}$$

- **Directional components**:

$$u_s = u_{s,\text{mag}} \cos(\theta_w \pi / 180) \tag{2}$$
$$v_s = u_{s,\text{mag}} \sin(\theta_w \pi / 180) \tag{3}$$

If $H_w \leq 0$ or $T_w \leq 0$, the method returns $(u_s, v_s) = (0, 0)$. NaN or Infinity values are clamped to zero to ensure numerical stability.

Wave-Enhanced Friction

The `ComputeWaveEnhancedFriction` method calculates an enhanced bottom friction coefficient:

- **Wave orbital velocity** at the bottom:

$$u_b = \frac{a\omega}{\sinh(kh)} \tag{4}$$

  where $a = H_w/2$, $\omega = 2\pi/T_w$, $k = 2\pi/\sqrt{gh}T_w$.

- **Enhanced friction coefficient**:

$$C_d = C_{d0}(1 + \beta|u_b|) \tag{5}$$

  where C_{d0} is the base friction coefficient, and $\beta = 0.2$ is an empirical factor.

- **Constraints**: The result is clamped between C_{d0} and 0.01 to prevent numerical instability.

If $H_w \leq 0$ or $T_w \leq 0$, the base friction coefficient is returned unchanged.

Physical and Mathematical Models

The `WaveCurrentInteraction` class employs the following models:

- **Stokes Drift**:

$$\lambda = \sqrt{gh}T_w \tag{6}$$
$$k = \frac{2\pi}{\lambda}, \quad \omega = \frac{2\pi}{T_w}, \quad a = \frac{H_w}{2} \tag{7}$$
$$u_{s,\text{mag}} = \frac{a^2 \omega k}{2\sinh^2(kh)} \cosh(2k \cdot 0) \tag{8}$$
$$u_s = u_{s,\text{mag}} \cos\left(\theta_w \frac{\pi}{180}\right), \quad v_s = u_{s,\text{mag}} \sin\left(\theta_w \frac{\pi}{180}\right) \tag{9}$$

- **Wave-Enhanced Friction**:

$$u_b = \frac{a\omega}{\sinh(kh)} \tag{10}$$
$$C_d = C_{d0}(1 + 0.2|u_b|), \quad C_d \in [C_{d0}, 0.01] \tag{11}$$

These models capture wave-induced transport and turbulence effects, integrating with estuarine circulation models to enhance realism in velocity and friction calculations.

Wind Forcing

Core Components and Initialization

The `WindForcing` class, implemented in C#, models wind-induced stress on an estuarine surface, incorporating the effects of wind speed, direction, and wave height. The class is initialized with the following parameters:

- Air density: $\rho_{\text{air}} = 1.225\,\text{kg/m}^3$
- Wind speed at 10 m height: U_{10} (m/s, non-negative)
- Wind direction: θ_w (degrees, normalized to $[0, 360)$)
- Significant wave height: H_s (m, non-negative)

The constructor enforces $U_{10} \geq 0$, $H_s \geq 0$, and normalizes θ_w using modulo 360. The `UpdateParameters` method allows dynamic updates of these parameters during simulation.

Functioning Logic

The class provides two primary methods to compute wind effects:

1. `ComputeDragCoefficient`: Calculates the wind drag coefficient based on wind speed and wave height.
2. `ComputeWindStress`: Computes wind stress components (τ_x, τ_y) in N/m², resolved along the x- and y-axes.

Drag Coefficient Computation

The `ComputeDragCoefficient` method uses a simplified parameterization:

$$C_d = (0.75 + 0.067U_{10} + 0.1H_s) \times 10^{-3} \tag{1}$$

The drag coefficient is bounded for physical realism:

$$C_d \in [0.001, 0.003] \tag{2}$$

This accounts for the influence of wind speed and sea state (via wave height) on surface drag.

Wind Stress Computation

The `ComputeWindStress` method calculates the wind stress components:

$$\tau_x = \rho_{\text{air}} C_d U_{10}^2 \cos\left(\theta_w \frac{\pi}{180}\right) \tag{3}$$

$$\tau_y = \rho_{\text{air}} C_d U_{10}^2 \sin\left(\theta_w \frac{\pi}{180}\right) \tag{4}$$

where C_d is obtained from `ComputeDragCoefficient`. The stress components represent the force per unit area exerted by the wind on the water surface, resolved in the x- and y-directions based on the wind direction.

Physical and Mathematical Models

The `WindForcing` class employs the following models:

- **Drag Coefficient**:

$$C_d = (0.75 + 0.067U_{10} + 0.1H_s) \times 10^{-3}, \quad C_d \in [0.001, 0.003] \tag{5}$$

- **Wind Stress**:

$$\tau_x = \rho_{\text{air}} C_d U_{10}^2 \cos\left(\theta_w \frac{\pi}{180}\right) \tag{6}$$

$$\tau_y = \rho_{\text{air}} C_d U_{10}^2 \sin\left(\theta_w \frac{\pi}{180}\right) \tag{7}$$

These models capture the wind-induced momentum transfer to the estuarine surface, accounting for variations in wind speed, direction, and sea state, and are designed for integration with broader estuarine circulation models.

Wet and Dry Algorithm

Core Components and Initialization

The `WetAndDryAlgo` class, implemented in C#, manages wetting and drying processes in a 2D estuarine circulation model, enabling simulation of areas that transition between inundated (wet) and exposed (dry) states. The class is initialized with the following parameters:

- Bathymetry: $h_b(i, j)$ (m, positive downward)
- Minimum depth threshold: D_{min} (m)
- Grid dimensions: n_x, n_y
- Estuary dimensions: length L, width W
- Spatial steps: $\Delta x = L/(n_x - 1)$, $\Delta y = W/(n_y - 1)$

The constructor initializes a boolean array `isWet` to track wet/dry status and calls `InitializeWetDry` with a zero water level to set initial conditions.

Functioning Logic

The class provides methods to handle wetting and drying:

1. `InitializeWetDry`: Sets the initial wet/dry status based on total water depth.
2. `ApplyWetDry`: Updates wet/dry status, enforces zero velocities and salinity in dry cells, adjusts fluxes at wet/dry interfaces, and applies mass conservation corrections.
3. `GetWetDryStatus`: Returns the current wet/dry status array.

Initialization

The `InitializeWetDry` method determines the wet/dry status for each grid cell:

$$\text{isWet}[i, j] = \eta_{i,j} + h_b(i, j) \geq D_{\text{min}} \tag{1}$$

where $\eta_{i,j}$ is the water level (positive upward) and $h_b(i, j)$ is the bathymetry (positive downward). A cell is wet if the total depth $\eta_{i,j} + h_b(i, j)$ exceeds the minimum depth threshold D_{min}.

Wet/Dry Application

The `ApplyWetDry` method updates the simulation state:

1. **Wet/Dry Status Update**: For each cell:

$$\text{isWet}[i, j] = \eta_{i,j} + h_b(i, j) \geq D_{\text{min}} \tag{2}$$

If a cell is dry (isWet$[i, j] =$ false):

$$u_{i,j} = 0, \quad v_{i,j} = 0 \tag{3}$$

$$\eta_{i,j} = -h_b(i, j) \quad \text{(water level set to bed level)} \tag{4}$$

$$S_{i,j} = 0 \quad \text{(no salinity in dry cells)} \tag{5}$$

2. **Flux Adjustment**: At wet/dry interfaces, velocities are modified to prevent unphysical flow:
   - If cell $(i + 1, j)$ is dry: $u_{i,j} = \min(u_{i,j}, 0)$ (prevent flow into dry cell).
   - If cell $(i - 1, j)$ is dry: $u_{i-1,j} = \max(u_{i-1,j}, 0)$ (prevent flow from dry cell).
   - Similarly for v in the y-direction.
3. **Mass Conservation Correction**: For wet cells, water level is adjusted based on net flux:

$$\text{fluxIn} = \begin{cases} u_{i-1,j}(\eta_{i-1,j} + h_b(i-1, j))\Delta y \Delta t & \text{if isWet}[i-1, j] \\ v_{i,j-1}(\eta_{i,j-1} + h_b(i, j-1))\Delta x \Delta t & \text{if isWet}[i, j-1] \end{cases} \tag{6}$$

$$\text{fluxOut} = \begin{cases} u_{i,j}(\eta_{i,j} + h_b(i, j))\Delta y \Delta t & \text{if isWet}[i+1, j] \\ v_{i,j}(\eta_{i,j} + h_b(i, j))\Delta x \Delta t & \text{if isWet}[i, j+1] \end{cases} \tag{7}$$

$$\eta_{i,j} \leftarrow \eta_{i,j} + \frac{\text{fluxIn} - \text{fluxOut}}{\Delta x \Delta y} \tag{8}$$

$$\eta_{i,j} = \max(-h_b(i, j), \eta_{i,j}) \quad \text{(ensure no negative depth)} \tag{9}$$

Physical and Mathematical Models

The `WetAndDryAlgo` class employs the following models:

- **Wet/Dry Condition**:

$$\text{isWet}[i, j] = \eta_{i,j} + h_b(i, j) \geq D_{\text{min}} \tag{10}$$

- **Dry Cell Conditions**:

$$u_{i,j} = v_{i,j} = S_{i,j} = 0, \quad \eta_{i,j} = -h_b(i, j) \tag{11}$$

- **Flux Adjustment**:

$$u_{i,j} = \min(u_{i,j}, 0) \quad \text{if } \neg\text{isWet}[i+1, j] \tag{12}$$

$$u_{i-1,j} = \max(u_{i-1,j}, 0) \quad \text{if } \neg\text{isWet}[i-1, j] \tag{13}$$

$$v_{i,j} = \min(v_{i,j}, 0) \quad \text{if } \neg\text{isWet}[i, j+1] \tag{14}$$

$$v_{i,j-1} = \max(v_{i,j-1}, 0) \quad \text{if } \neg\text{isWet}[i, j-1] \tag{15}$$

- **Mass Conservation**:

$$\eta_{i,j} \leftarrow \eta_{i,j} + \frac{\sum \text{fluxIn} - \sum \text{fluxOut}}{\Delta x \Delta y}, \quad \eta_{i,j} \geq -h_b(i, j) \tag{16}$$

These models ensure physically consistent transitions between wet and dry states, prevent unphysical flows, and maintain mass conservation in the 2D shallow water model.

Simpson-Hunter Mechanism Parameterization

Core Components and Initialization

The `SimpsonHunterMechanismParam` class, implemented in C#, models physical processes related to tidal straining and internal tides in estuarine circulation, focusing on wave-induced Stokes drift, internal tide effects, and tidal straining (Simpson-Hunter mechanism). The class is initialized with the following parameters:

- Number of sigma layers: n_σ (vertical grid points)
- Wave amplitude: $a = 0.5\,\text{m}$ (default)
- Wave period: $T_w = 10.0\,\text{s}$ (default)
- Internal tide amplitude: $A_{\text{it}} = 0.05\,\text{m/s}$ (default)
- Simpson-Hunter coefficient: $C_{\text{sh}} = 0.1$ (default, for tidal straining)
- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Reference density: $\rho_0 = 1000.0\,\text{kg/m}^3$

The constructor sets these parameters, allowing customization of wave, internal tide, and tidal straining effects.

Functioning Logic

The class provides three methods to compute contributions to estuarine dynamics:

1. `ComputeStokesDrift`: Calculates the x-direction Stokes drift velocity due to surface waves.
2. `ComputeInternalTideEffect`: Computes vertical velocity perturbations and turbulent kinetic energy (TKE) production from internal tides.
3. `ComputeTidalStraining`: Evaluates the tidal straining effect (Simpson-Hunter mechanism) on the salinity gradient.

Stokes Drift Computation

The `ComputeStokesDrift` method calculates the Stokes drift velocity in the x-direction using a simplified shallow-water approximation:

- Wave number: $k = \frac{2\pi}{T_w \sqrt{gh}}$, where h is the water depth.
- Depth coordinate: $z = \sigma h$, where σ is the sigma coordinate (0 at bottom, 1 at surface).
- Stokes drift velocity:

$$u_s = a \cdot \frac{2\pi}{T_w} \cdot e^{-2kz} \cdot \sin(\phi_t) \tag{1}$$

where a is the wave amplitude, ϕ_t is the tidal phase, and the velocity is capped: $u_s \in [-0.5, 0.5]\,\text{m/s}$ for numerical stability.

Internal Tide Effect

The `ComputeInternalTideEffect` method computes vertical velocity perturbations and TKE production due to internal tides:

- **Buoyancy frequency squared**:

$$N^2 = -\frac{g}{\rho_0}\frac{\partial \rho}{\partial z}, \quad \frac{\partial \rho}{\partial z} = \frac{\rho_2 - \rho_1}{\Delta\sigma \cdot h} \tag{2}$$

  where $\rho_k = \rho_0 + 0.8 S_k$, and S_k is the salinity at layer k. N^2 is capped: $N^2 \in [0, 10^{-3}]\,\text{s}^{-2}$.

- **Vertical velocity perturbation**:

$$w_{\text{tide}} = A_{\text{it}} \sin(\phi_t)\sqrt{N^2}\cos\left(\frac{2\pi z}{h}\right) \tag{3}$$

  where $w_{\text{tide}} \in [-0.2, 0.2]\,\text{m/s}$.

- **TKE production**:

$$\text{TKE} = 0.1 A_{\text{it}} \left(\frac{\partial w}{\partial z}\right)^2, \quad \frac{\partial w}{\partial z} = \frac{w_k - w_{k+1}}{\Delta\sigma \cdot h} \tag{4}$$

  TKE is capped: $\text{TKE} \in [0, 10^{-4}]\,\text{m}^2/\text{s}^3$. No effect is applied at the top layer ($k = n_\sigma - 1$).

Tidal Straining Effect

The `ComputeTidalStraining` method models the Simpson-Hunter mechanism, which describes how tidal currents modify salinity gradients:

- **Salinity gradient**:

$$\frac{\partial S}{\partial x} = \begin{cases} \frac{S_{i+1,k} - S_{i-1,k}}{2\Delta x} & \text{if } 0 < i < n_{\text{cells}} - 1 \\ \frac{S_{i+1,k} - S_{i,k}}{\Delta x} & \text{if } i = 0 \\ \frac{S_{i,k} - S_{i-1,k}}{\Delta x} & \text{if } i = n_{\text{cells}} - 1 \end{cases} \tag{5}$$

  where Δx is the average cell width, and $\frac{\partial S}{\partial x} \in [-10, 10]\,\text{PSU/m}$.

- **Tidal straining term**:

$$\text{Straining} = -C_{\text{sh}} u_t \frac{\partial S}{\partial x} \tag{6}$$

  where u_t is the tidal velocity, and the term is capped: $\text{Straining} \in [-1, 1]\,\text{PSU/s}$. No effect is applied at the top layer or boundary cells.

Physical and Mathematical Models

The `SimpsonHunterMechanismParam` class employs the following models:

- **Stokes Drift**:

$$u_s = a \cdot \frac{2\pi}{T_w} \cdot e^{-2kz} \cdot \sin(\phi_t), \quad u_s \in [-0.5, 0.5] \tag{7}$$

- **Internal Tide**:

$$N^2 = -\frac{g}{\rho_0} \frac{\rho_2 - \rho_1}{\Delta\sigma \cdot h}, \quad N^2 \in [0, 10^{-3}] \tag{8}$$

$$w_{\text{tide}} = A_{\text{it}} \sin(\phi_t) \sqrt{N^2} \cos\left(\frac{2\pi z}{h}\right), \quad w_{\text{tide}} \in [-0.2, 0.2] \tag{9}$$

$$\text{TKE} = 0.1 A_{\text{it}} \left(\frac{w_k - w_{k+1}}{\Delta\sigma \cdot h}\right)^2, \quad \text{TKE} \in [0, 10^{-4}] \tag{10}$$

- **Tidal Straining**:

$$\text{Straining} = -C_{\text{sh}} u_t \frac{\partial S}{\partial x}, \quad \frac{\partial S}{\partial x} \in [-10, 10], \quad \text{Straining} \in [-1, 1] \tag{11}$$

These models capture wave-induced transport, internal tide-driven vertical mixing, and tidal straining effects on stratification, enhancing the realism of estuarine circulation simulations.

Asymmetric Tidal Mixing

Core Components and Initialization

The `AsymmTidalMix` class, implemented in C#, simulates asymmetric tidal mixing in a bifurcated estuarine system using a 3D hydrodynamic solver on an unstructured grid with sigma coordinates. It integrates physical processes such as tidal forcing, Stokes drift, internal tides, tidal straining, and turbulence using a $k-\epsilon$ model. The class is initialized with the following parameters:

- Estuary length: $L = 10000\,\text{m}$
- Number of cells: $n_{\text{cells}} = 70$ (30 main channel, 20 branch 1, 20 branch 2)
- Number of sigma layers: $n_\sigma = 10$
- Time step: $\Delta t = 100\,\text{s}$
- Kinematic viscosity: $\nu = 10^{-6}\,\text{m}^2/\text{s}$
- Gravitational acceleration: $g = 9.81\,\text{m}/\text{s}^2$
- Reference density: $\rho_0 = 1000\,\text{kg}/\text{m}^3$
- Coriolis parameter: $f = 10^{-4}\,\text{s}^{-1}$
- Tidal amplitude: $A_t = 1.0\,\text{m}/\text{s}$
- Default bed shear stress: $\tau_b = 0.1\,\text{N}/\text{m}^2$
- Default tidal period: $T_t = 43200\,\text{s}$ (12 hours)

The class uses a Windows Forms interface for user interaction and visualization, incorporating models for mixing (`RichardsonNumDepAndSSIMix`), tidal effects (`SimpsonHunterMechanismParam`), and bifurcated estuary geometry (`BifurcatedEstuaryM`
Each cell in the unstructured grid is defined by the `Cell` class, storing:

- X-coordinate: X (m)
- Depth: h (m)
- Volume: $V = Ah$ (m^3, where A is the cell area)
- Arrays per sigma layer: turbulent kinetic energy (K), dissipation rate (ϵ), velocities (u, v, w), pressure (p), salinity (S), turbidity, shear
- Neighbor indices for connectivity

Initial conditions include a linear salinity gradient ($S = 35(1 - x/L)$ PSU), $K = 10^{-4}\,\text{m}^2/\text{s}^2$, and $\epsilon = 10^{-6}\,\text{m}^2/\text{s}^3$.

Functioning Logic

The `AsymmTidalMix` class manages:

1. **User Interface**: A Windows Forms window with controls for bed shear stress (τ_b), tidal period (T_t), visualization options (velocity vectors, streamlines, salinity isosurface, density slice, turbidity fronts, shear layer), and simulation controls (start, pause, reset).
2. **Grid Initialization**: Uses `BifurcatedEstuaryModels` to create an unstructured grid with 70 cells, representing a main channel and two branches.
3. **Simulation Update**: Advances the simulation using a 3D Navier-Stokes solver with Boussinesq approximation, incorporating:
   - Tidal forcing with asymmetry (flood: 1.2, ebb: 0.8)
   - Stokes drift and internal tide effects via `SimpsonHunterMechanismParam`
   - Turbulent viscosity via `RichardsonNumDepAndSSIMix`
   - Bifurcated dynamics via `BifurcatedEstuaryModels`
4. **Turbulence Modeling**: Updates K and ϵ using a $k-\epsilon$ model with contributions from shear, internal tides, and bifurcated mixing.
5. **Visualization**: Displays profiles, isosurfaces, and hodographs in two panels.
6. **Output Console**: Reports average TKE, velocity, salinity, turbidity, shear, Richardson number, and tidal phase.

Simulation Update

The `UpdateSimulation` method advances the simulation by $\Delta t = 100\,\text{s}$, updating the tidal phase:

$$\phi_t = \frac{2\pi t}{T_t} \tag{1}$$

The simulation follows these steps:

1. **Intermediate Velocities**: Solve Navier-Stokes equations:

$$\frac{u^* - u^n}{\Delta t} = -u\frac{\partial u}{\partial x} + (\nu + \nu_T)\nabla^2 u + fv - \frac{\tau_b}{\rho_0}\delta_{k=0}(u + u_s) \tag{2}$$

$$\frac{v^* - v^n}{\Delta t} = -u\frac{\partial v}{\partial x} + (\nu + \nu_T)\nabla^2 v - f(u + u_s) - \frac{\tau_b}{\rho_0}\delta_{k=0} v \tag{3}$$

$$\frac{w^* - w^n}{\Delta t} = -w\frac{\partial w}{\partial z} + (\nu + \nu_T)\nabla^2 w - \frac{g(\rho - \rho_0)}{\rho_0} + w_{\text{tide}} \tag{4}$$

   where u_s is Stokes drift, w_{tide} is internal tide vertical velocity, ν_T is turbulent viscosity, and $\rho = \rho_0 + 0.8S$. Boundary conditions:

   - Ocean ($i = 0$): $u = A_t \sin(\phi_t)(1 - \sigma)$, $v = w = 0$, $S = 35\,\text{PSU}$
   - Branch ends ($i = 49, 69$): $u = 0.1(1 - \sigma) + A_t'(1 - \sigma)$, $v = w = 0$, $S = 0\,\text{PSU}$
   - Bed ($k = 0$): No-slip ($u = v = w = 0$)

   Velocities are capped: $u, v, w \in [-2, 2]\,\text{m/s}$.

2. **Pressure Poisson Equation**: Solve for pressure to enforce incompressibility:
$$\nabla^2 p = \frac{\rho_0}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{5}$$
using successive over-relaxation (SOR) with 20 iterations. Pressure is capped: $p \in [-10^5, 10^5]$ Pa.

3. **Velocity Correction**:
$$u^{n+1} = u^* - \frac{\Delta t}{\rho_0} \frac{\partial p}{\partial x} \tag{6}$$
$$v^{n+1} = v^* \tag{7}$$
$$w^{n+1} = w^* - \frac{\Delta t}{\rho_0} \frac{\partial p}{\partial z} \tag{8}$$

4. **Turbulence Update**: Solve $k - \epsilon$ equations:
$$\frac{k^{n+1} - k^n}{\Delta t} = -u \frac{\partial k}{\partial x} + \frac{\partial}{\partial x} \left(\frac{\nu + \nu_T}{\sigma_k} \frac{\partial k}{\partial x} \right) + P_k + P_{\text{it}} - \epsilon \tag{9}$$
$$\frac{\epsilon^{n+1} - \epsilon^n}{\Delta t} = -u \frac{\partial \epsilon}{\partial x} + \frac{\partial}{\partial x} \left(\frac{\nu + \nu_T}{\sigma_\epsilon} \frac{\partial \epsilon}{\partial x} \right) + C_{1\epsilon} \frac{\epsilon}{k} P_k - C_{2\epsilon} \frac{\epsilon^2}{k} \tag{10}$$
where P_k is shear production, P_{it} is internal tide TKE production, $\sigma_k = 1.0$, $\sigma_\epsilon = 1.3$, $C_{1\epsilon} = 1.44$, $C_{2\epsilon} = 1.92$. $k \in [10^{-6}, 10^{-3}]$ m^2/s^2, $\epsilon \in [10^{-8}, 10^{-5}]$ m^2/s^3.

5. **Salinity and Turbidity**: Update using advection-diffusion with tidal straining:
$$\frac{S^{n+1} - S^n}{\Delta t} = -u \frac{\partial S}{\partial x} + \frac{\partial}{\partial z} \left(\frac{\nu + \nu_T}{\sigma_S} \frac{\partial S}{\partial z} \right) + S_{\text{strain}} \tag{11}$$
$$\frac{T^{n+1} - T^n}{\Delta t} = -u \frac{\partial T}{\partial x} + \frac{\partial}{\partial z} \left(\frac{\nu + \nu_T}{\sigma_T} \frac{\partial T}{\partial z} \right) + P_T - \frac{T}{1000} \tag{12}$$
where S_{strain} is the tidal straining term, $P_T = 0.5k$ at the bed, $\sigma_S = \sigma_T = 1.0$. Salinity is capped: $S \in [0, 35]$ PSU; turbidity: $T \in [0, 100]$.

6. **Shear Calculation**:
$$\text{Shear} = \sqrt{\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2}, \quad \text{Shear} \in [0, 10]\,\text{s}^{-1} \tag{13}$$

Visualization

The `VisualizationPanel_Paint` method renders:

- **Density Slice**: Color-coded density ($\rho = \rho_0 + 0.8S$) across cells and sigma layers (blue gradient).
- **Salinity Isosurface**: Lines at $S = 15$ PSU (green, thicker during flood).
- **Shear Layer**: Lines scaled by shear magnitude (orange, thicker during flood).

- **Turbidity Fronts**: Lines at 50% of maximum turbidity (brown, thicker during flood).
- **Velocity Vectors**: Arrows showing u, w at selected cells and layers (black, thicker during flood).
- **Streamlines**: Paths following average u velocity (purple, thicker during flood).
- **Turbulence and Velocity Profiles**: Blue (TKE) and red (velocity magnitude) lines along the estuary.

Values are scaled to panel dimensions, with checks for NaN/Infinity and bounds enforcement.

The `HodographPanel_Paint` method plots hodographs for u, w velocities at three sample points (main channel, branch 1, branch 2) in blue, green, and red, respectively, updated every $T_t/12$.

Physical and Mathematical Models

The `AsymmTidalMix` class employs:

- **Navier-Stokes with Boussinesq**:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho_0}\nabla p + (\nu + \nu_T)\nabla^2 \mathbf{u} + \mathbf{f} \times \mathbf{u} - \frac{g}{\rho_0}\nabla \rho + \mathbf{F} \tag{14}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{15}$$

  where **F** includes bed friction, Stokes drift, and internal tides.

- **Turbulence ($k - \epsilon$)**:

$$\frac{\partial k}{\partial t} + u\frac{\partial k}{\partial x} = \frac{\partial}{\partial x}\left(\frac{\nu + \nu_T}{\sigma_k}\frac{\partial k}{\partial x}\right) + P_k + P_{\text{it}} - \epsilon \tag{16}$$

$$\frac{\partial \epsilon}{\partial t} + u\frac{\partial \epsilon}{\partial x} = \frac{\partial}{\partial x}\left(\frac{\nu + \nu_T}{\sigma_\epsilon}\frac{\partial \epsilon}{\partial x}\right) + C_{1\epsilon}\frac{\epsilon}{k}P_k - C_{2\epsilon}\frac{\epsilon^2}{k} \tag{17}$$

- **Salinity and Turbidity**:

$$\frac{\partial S}{\partial t} + u\frac{\partial S}{\partial x} = \frac{\partial}{\partial z}\left(\frac{\nu + \nu_T}{\sigma_S}\frac{\partial S}{\partial z}\right) + S_{\text{strain}} \tag{18}$$

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = \frac{\partial}{\partial z}\left(\frac{\nu + \nu_T}{\sigma_T}\frac{\partial T}{\partial z}\right) + P_T - \frac{T}{1000} \tag{19}$$

- **Boundary Conditions**:
  - Ocean: Tidal velocity and salinity
  - Branch ends: Freshwater inflow and scaled tidal velocity
  - Bed: No-slip and bed friction

The model integrates tidal asymmetry, turbulence, and estuarine geometry to simulate realistic circulation and mixing dynamics, with interactive visualization and user-configurable parameters.

Bifurcated Estuary Models

Core Components and Initialization

The `BifurcatedEstuaryModels` class, implemented in C#, defines a bifurcated estuarine grid with a main channel and two branches, enabling simulation of spatially varying dynamics in estuarine circulation models. It integrates with the `AsymmTidalMix` class to provide grid initialization and branch-specific adjustments for tidal velocity and turbulent mixing. The class is initialized with the following parameters:

- Number of cells in main channel: $n_{\text{main}} = 30$
- Number of cells per branch: $n_{\text{branch}} = 20$
- Estuary length: $L = 10000\,\text{m}$
- Bifurcation point: $x_b = 0.6L$ (fraction of estuary length)
- Depth scaling for branch 1: $s_{d1} = 0.8$ (shallower)
- Depth scaling for branch 2: $s_{d2} = 1.2$ (deeper)
- Tidal velocity scaling for branch 1: $s_{t1} = 0.7$ (reduced tidal influence)
- Tidal velocity scaling for branch 2: $s_{t2} = 1.3$ (enhanced tidal influence)

The constructor sets these parameters, which control the geometry and dynamics of the bifurcated estuary.

Functioning Logic

The class provides three main methods to support estuarine modeling:

1. `InitializeBifurcatedGrid`: Creates an unstructured grid with cells for the main channel and two branches, assigning coordinates, depths, areas, and neighbor connectivity.
2. `GetBranchIndex`: Identifies whether a cell belongs to the main channel or one of the branches.
3. `AdjustMixingDynamics`: Modifies turbulent viscosity and turbulent kinetic energy (TKE) production based on branch-specific depth and tidal scaling.
4. `AdjustTidalVelocity`: Scales tidal velocity at branch ends to reflect differing tidal influences.

Grid Initialization

The `InitializeBifurcatedGrid` method constructs a grid with $n_{\text{main}} + 2 \cdot n_{\text{branch}} = 70$ cells:

- **Main Channel**: $n_{\text{main}} = 30$ cells from $x = 0$ (ocean) to $x = x_b = 0.6L$.

  - Cell width: $\Delta x_{\text{main}} = \frac{x_b}{n_{\text{main}}}$
  - X-coordinate: $x_i = i\Delta x_{\text{main}} + \text{rand} \cdot 0.5\Delta x_{\text{main}}$, where rand $\in [0, 1)$.
  - Depth: $h_i = 5.0 + 5.0 \sin\left(\frac{\pi x_i}{L}\right)$ m
  - Area: $A_i = \Delta x_{\text{main}} \cdot 100 \cdot (0.8 + \text{rand} \cdot 0.4)$ m^2
- **Branch 1**: $n_{\text{branch}} = 20$ cells from $x = x_b$ to $x = L$.
  - Cell width: $\Delta x_{\text{branch}} = \frac{L - x_b}{n_{\text{branch}}}$
  - X-coordinate: $x_i = x_b + i\Delta x_{\text{branch}} + \text{rand} \cdot 0.5\Delta x_{\text{branch}}$
  - Depth: $h_i = \left[5.0 + 5.0 \sin\left(\frac{\pi x_i}{L}\right)\right] \cdot s_{d1}$ m
  - Area: $A_i = \Delta x_{\text{branch}} \cdot 100 \cdot (0.8 + \text{rand} \cdot 0.4)$ m^2
- **Branch 2**: $n_{\text{branch}} = 20$ cells from $x = x_b$ to $x = L$.
  - Same cell width and x-coordinate as Branch 1
  - Depth: $h_i = \left[5.0 + 5.0 \sin\left(\frac{\pi x_i}{L}\right)\right] \cdot s_{d2}$ m
  - Area: Same as Branch 1
- **Neighbor Connectivity**:
  - Main channel: Linear (cell i connects to $i - 1$ and $i + 1$), with the last cell ($i = n_{\text{main}} - 1$) connecting to the first cells of both branches.
  - Branch 1: Linear (cell i connects to $i - 1$, $i + 1$), with the first cell connecting to the bifurcation point.
  - Branch 2: Similar to Branch 1.

A fixed random seed ensures reproducibility.

Branch Identification

The `GetBranchIndex` method assigns a cell to a region:

$$\text{Branch Index} = \begin{cases} 0 & \text{if } i < n_{\text{main}} \text{ (main channel)} \\ 1 & \text{if } n_{\text{main}} \leq i < n_{\text{main}} + n_{\text{branch}} \text{ (branch 1)} \\ 2 & \text{if } i \geq n_{\text{main}} + n_{\text{branch}} \text{ (branch 2)} \end{cases} \tag{1}$$

Mixing Dynamics Adjustment

The `AdjustMixingDynamics` method modifies turbulent viscosity (ν_T) and TKE production (P_k) based on the branch:

- **Mixing Modifier**:

$$m = \begin{cases} 1.0 & \text{(main channel)} \\ s_{d1} = 0.8 & \text{(branch 1, reduced mixing)} \\ s_{d2} = 1.2 & \text{(branch 2, enhanced mixing)} \end{cases} \tag{2}$$

- **Turbulent Viscosity**: $\nu_T \leftarrow \nu_T \cdot m$
- **TKE Production**:

$$P_k \leftarrow P_k + \nu_T \cdot \text{Shear}_k^2 \cdot m \cdot \frac{u_t \cdot s_t}{\max(0.01, |u_t|)} \tag{3}$$

where u_t is the tidal velocity, $s_t = s_{t1}$ or s_{t2} for branches 1 and 2, respectively, and $P_k \in [0, 10^{-3}]\,\text{m}^2/\text{s}^3$.

Tidal Velocity Adjustment

The `AdjustTidalVelocity` method scales the tidal velocity at branch ends:

$$u_t' = \begin{cases} u_t \cdot s_{t1} & \text{(branch 1, cell } i = n_{\text{main}} + n_{\text{branch}} - 1\text{)} \\ u_t \cdot s_{t2} & \text{(branch 2, cell } i = n_{\text{main}} + 2 \cdot n_{\text{branch}} - 1\text{)} \\ u_t & \text{(main channel)} \end{cases} \tag{4}$$

Physical and Mathematical Models

The `BifurcatedEstuaryModels` class employs the following models:

- **Grid Geometry**:

$$x_i = i\Delta x + \text{rand} \cdot 0.5\Delta x \tag{5}$$

$$h_i = \left(5.0 + 5.0 \sin\left(\frac{\pi x_i}{L}\right)\right) \cdot s_d, \quad s_d = \begin{cases} 1.0 & \text{(main channel)} \\ s_{d1} & \text{(branch 1)} \\ s_{d2} & \text{(branch 2)} \end{cases} \tag{6}$$

$$A_i = \Delta x \cdot 100 \cdot (0.8 + \text{rand} \cdot 0.4) \tag{7}$$

- **Turbulent Viscosity Adjustment**:

$$\nu_T \leftarrow \nu_T \cdot m, \quad m = \begin{cases} 1.0 & \text{(main channel)} \\ 0.8 & \text{(branch 1)} \\ 1.2 & \text{(branch 2)} \end{cases} \tag{8}$$

- **TKE Production Adjustment**:

$$P_k \leftarrow P_k + \nu_T \cdot \text{Shear}_k^2 \cdot m \cdot \frac{u_t \cdot s_t}{\max(0.01, |u_t|)}, \quad P_k \in [0, 10^{-3}] \tag{9}$$

- **Tidal Velocity Scaling**:

$$u_t' = u_t \cdot s_t, \quad s_t = \begin{cases} 0.7 & \text{(branch 1)} \\ 1.3 & \text{(branch 2)} \\ 1.0 & \text{(main channel)} \end{cases} \tag{10}$$

These models capture the geometric and dynamic variations in a bifurcated estuary, enabling realistic simulation of tidal and mixing processes in distinct channels.

Equation of State

Core Components and Initialization

The `EqOfState` class, implemented in C#, calculates seawater density based on salinity, temperature, and pressure, using a polynomial approximation suitable for estuarine circulation modeling. This static class provides a single method, `ComputeDensity`, which is integrated into the `AsymmTidalMix` framework to determine density-driven dynamics in estuarine environments.

Functioning Logic

The `ComputeDensity` method computes seawater density (ρ) in kg/m³, taking:

- Salinity (S): Practical Salinity Units (PSU)
- Temperature (T): Degrees Celsius (°C)
- Pressure (P): Decibars (dbar, where 1 dbar = 10^4 Pa)

Input parameters are constrained to physical ranges to ensure numerical stability:

- $S \in [0, 40]$ PSU
- $T \in [-2, 40]$ °C
- $P \geq 0$ dbar

Density Computation

The method uses a polynomial equation of state to compute density, incorporating contributions from pure water, salinity, temperature-salinity interactions, and pressure effects via compressibility. The computation proceeds in steps:

1. **Pure Water Density** (ρ_0) at zero pressure:

$$\rho_0 = a_0 + a_1 T + a_2 T^2 + a_3 T^3 + a_4 T^4 + a_5 T^5 \tag{1}$$

   where coefficients are:

   - $a_0 = 999.842594$
   - $a_1 = 6.793952 \times 10^{-2}$
   - $a_2 = -9.095290 \times 10^{-3}$
   - $a_3 = 1.001685 \times 10^{-4}$
   - $a_4 = -1.120083 \times 10^{-6}$
   - $a_5 = 6.536332 \times 10^{-9}$

2. **Salinity Contribution**:

$$S_{\text{term}} = b_0 + b_1 T + b_2 T^2 + b_3 T^3 + b_4 T^4 \tag{2}$$

where coefficients are:

- $b_0 = 8.24493 \times 10^{-1}$
- $b_1 = -4.0899 \times 10^{-3}$
- $b_2 = 7.6438 \times 10^{-5}$
- $b_3 = -8.2467 \times 10^{-7}$
- $b_4 = 5.3875 \times 10^{-9}$

3. **Temperature-Salinity Interaction**:

$$I_{\text{term}} = c_0 + c_1 T + c_2 T^2 \tag{3}$$

where coefficients are:

- $c_0 = -5.72466 \times 10^{-3}$
- $c_1 = 1.0227 \times 10^{-4}$
- $c_2 = -1.6546 \times 10^{-6}$

4. **Salinity $S^{1.5}$ Term**:

$$S_{\text{sqrt}} = d_0 S^{3/2}, \quad d_0 = 4.8314 \times 10^{-4} \tag{4}$$

5. **Density at Atmospheric Pressure**:

$$\rho = \rho_0 + S \cdot S_{\text{term}} + S \cdot I_{\text{term}} + S^{3/2} \cdot d_0 \tag{5}$$

6. **Compressibility (Secant Bulk Modulus)**:

$$K_0 = e_0 + e_1 T + e_2 T^2 + e_3 T^3 + e_4 T^4 \tag{6}$$
$$K_1 = f_0 + f_1 T + f_2 T^2 + f_3 T^3 \tag{7}$$
$$K_2 = g_0 + g_1 T + g_2 T^2 \tag{8}$$
$$K = K_0 + S \cdot K_1 + S \cdot S^{1/2} \cdot K_2 \tag{9}$$

where coefficients are:

- $e_0 = 19652.21$, $e_1 = 148.4206$, $e_2 = -2.327105$, $e_3 = 1.360477 \times 10^{-2}$, $e_4 = -5.155288 \times 10^{-5}$
- $f_0 = 54.6746$, $f_1 = -0.603459$, $f_2 = 1.09987 \times 10^{-2}$, $f_3 = -6.1670 \times 10^{-5}$
- $g_0 = 7.944 \times 10^{-2}$, $g_1 = 1.6483 \times 10^{-2}$, $g_2 = -5.3009 \times 10^{-4}$

7. **Pressure-Corrected Density**:

$$\rho = \rho \cdot \frac{1 + P/(K - P)}{1} \tag{10}$$

Physical and Mathematical Models

The `EqOfState` class employs the following model for seawater density:

- **Density at Atmospheric Pressure**:

$$\rho = \left(a_0 + \sum_{i=1}^{5} a_i T^i \right) + S \cdot \left(b_0 + \sum_{i=1}^{4} b_i T^i \right) + S \cdot \left(c_0 + c_1 T + c_2 T^2 \right) + d_0 S^{3/2} \quad (11)$$

- **Secant Bulk Modulus**:

$$K = \left(e_0 + \sum_{i=1}^{4} e_i T^i \right) + S \cdot \left(f_0 + \sum_{i=1}^{3} f_i T^i \right) + S \cdot S^{1/2} \cdot \left(g_0 + g_1 T + g_2 T^2 \right) \quad (12)$$

- **Pressure Correction**:

$$\rho \leftarrow \rho \cdot \left(1 + \frac{P}{K - P} \right) \quad (13)$$

This model provides accurate density calculations for estuarine waters, accounting for salinity, temperature, and pressure effects, with bounds ensuring numerical stability in simulations.

Vertical Discretization

Core Components and Initialization

The `VerticalDiscretization` abstract class, implemented in C#, provides a framework for defining vertical coordinate systems in estuarine circulation modeling. It supports the creation of computational grids for 2D or 3D hydrodynamic simulations, with derived classes implementing specific coordinate systems (sigma and z-level). The class is initialized with the following parameters:

- Number of grid points in x-direction: n_x (horizontal)
- Number of grid points in z-direction: n_z (vertical)
- Spatial step in x-direction: $\Delta\xi = \frac{L}{n_x - 1}$, where L is the estuary length (m)
- Spatial step in z-direction: $\Delta\eta$ (dimensionless for sigma, meters for z-level)
- Estuary length: L (m)
- Estuary depth: h (m)
- Horizontal coordinates: $x(i, j)$ (m, 2D array)

The class maintains arrays for:

- Vertical coordinates: $z(i, j)$ (m)
- Metric term: $\frac{\partial z}{\partial \xi}$
- Metric term: $\frac{\partial z}{\partial \eta}$

These arrays are accessible via properties `Z`, `Z_xi`, and `Z_eta`.

Functioning Logic

The `VerticalDiscretization` abstract class defines two abstract methods implemented by derived classes:

1. `InitializeGrid`: Sets up the vertical coordinate grid based on the chosen system.
2. `UpdateDepth`: Updates the grid when the estuary depth changes.

Two derived classes, `SigmaCoordinates` and `ZLevelCoordinates`, implement these methods for sigma and z-level coordinate systems, respectively.

Sigma Coordinates

The `SigmaCoordinates` class implements a terrain-following sigma coordinate system, where the vertical coordinate scales with the local water depth. It is initialized with the same parameters as the base class.

Grid Initialization

The `InitializeGrid` method defines the vertical coordinate:

$$z(i,j) = \eta_j \cdot h, \quad \eta_j = j \cdot \Delta\eta, \quad j = 0, 1, \ldots, n_z - 1 \tag{1}$$

where $\eta_j \in [0,1]$ is the sigma coordinate (0 at the bed, 1 at the surface), and h is the estuary depth. This ensures that grid layers follow the bathymetry.

Metric Terms

The method `ComputeMetricTerms` calculates spatial derivatives:

$$\frac{\partial z}{\partial \xi}(i,j) = \begin{cases} \frac{z(i+1,j)-z(i-1,j)}{2\Delta\xi} & \text{if } 0 < i < n_x - 1 \\ \frac{z(1,j)-z(0,j)}{\Delta\xi} & \text{if } i = 0 \\ \frac{z(n_x-1,j)-z(n_x-2,j)}{\Delta\xi} & \text{if } i = n_x - 1 \end{cases} \tag{2}$$

$$\frac{\partial z}{\partial \eta}(i,j) = \begin{cases} \frac{z(i,j+1)-z(i,j-1)}{2\Delta\eta} & \text{if } 0 < j < n_z - 1 \\ \frac{z(i,1)-z(i,0)}{\Delta\eta} & \text{if } j = 0 \\ \frac{z(i,n_z-1)-z(i,n_z-2)}{\Delta\eta} & \text{if } j = n_z - 1 \end{cases} \tag{3}$$

For sigma coordinates, since $z = \eta \cdot h$, and assuming constant depth across x, $\frac{\partial z}{\partial \xi} \approx 0$, while $\frac{\partial z}{\partial \eta} \approx h$.

Depth Update

The `UpdateDepth` method updates the estuary depth h and reinitializes the grid using the same sigma coordinate formulation.

Z-Level Coordinates

The `ZLevelCoordinates` class implements a z-level coordinate system with fixed horizontal layers, independent of bathymetry. It is initialized with the same parameters as the base class.

Grid Initialization

The `InitializeGrid` method defines the vertical coordinate:

$$z(i,j) = j \cdot \Delta z, \quad \Delta z = \frac{h}{n_z - 1}, \quad j = 0, 1, \ldots, n_z - 1 \tag{4}$$

where Δz is the uniform vertical spacing, and z ranges from 0 (bed) to h (surface).

Metric Terms

The `ComputeMetricTerms` method calculates the same derivatives as in sigma coordinates:

$$\frac{\partial z}{\partial \xi}(i,j) = \begin{cases} \frac{z(i+1,j)-z(i-1,j)}{2\Delta\xi} & \text{if } 0 < i < n_x - 1 \\ \frac{z(1,j)-z(0,j)}{\Delta\xi} & \text{if } i = 0 \\ \frac{z(n_x-1,j)-z(n_x-2,j)}{\Delta\xi} & \text{if } i = n_x - 1 \end{cases} \tag{5}$$

$$\frac{\partial z}{\partial \eta}(i,j) = \begin{cases} \frac{z(i,j+1)-z(i,j-1)}{2\Delta\eta} & \text{if } 0 < j < n_z - 1 \\ \frac{z(i,1)-z(i,0)}{\Delta\eta} & \text{if } j = 0 \\ \frac{z(i,n_z-1)-z(i,n_z-2)}{\Delta\eta} & \text{if } j = n_z - 1 \end{cases} \tag{6}$$

For z-level coordinates, since z is constant across x, $\frac{\partial z}{\partial \xi} = 0$, and $\frac{\partial z}{\partial \eta} = \frac{\Delta z}{\Delta \eta}$.

Depth Update

The `UpdateDepth` method updates the estuary depth h and reinitializes the grid with updated Δz.

Physical and Mathematical Models

The `VerticalDiscretization` framework supports two coordinate systems:

- **Sigma Coordinates**:

$$z(i,j) = \eta_j \cdot h, \quad \eta_j = j \cdot \Delta\eta, \quad \Delta\eta = \frac{1}{n_z - 1} \tag{7}$$

$$\frac{\partial z}{\partial \xi} \approx 0 \quad (\text{assuming constant depth}) \tag{8}$$

$$\frac{\partial z}{\partial \eta} \approx h \tag{9}$$

- **Z-Level Coordinates**:

$$z(i,j) = j \cdot \Delta z, \quad \Delta z = \frac{h}{n_z - 1} \tag{10}$$

$$\frac{\partial z}{\partial \xi} = 0 \tag{11}$$

$$\frac{\partial z}{\partial \eta} = \frac{\Delta z}{\Delta \eta} \tag{12}$$

These models provide flexible vertical discretization for estuarine modeling, with sigma coordinates suitable for terrain-following simulations and z-level coordinates for fixed-layer applications, ensuring accurate representation of vertical gradients and bathymetric variations.

Richardson Number Dependent and Shear-Strain-Induced Mixing

Core Components and Initialization

The `RichardsonNumDepAndSSIMix` class, implemented in C#, models turbulent mixing in estuarine circulation by computing Richardson number-dependent turbulent viscosity and shear-strain-induced turbulent kinetic energy (TKE) production. It integrates with the `AsymmTidalMix` class to enhance the realism of mixing processes. The class is initialized with:

- Number of sigma layers: n_σ (vertical grid points)
- Gravitational acceleration: $g = 9.81\,\text{m/s}^2$
- Reference density: $\rho_0 = 1000\,\text{kg/m}^3$
- Kinematic viscosity: $\nu = 10^{-6}\,\text{m}^2/\text{s}$
- $k - \epsilon$ model constant: $c_\mu = 0.09$

Functioning Logic

The class provides three methods to compute mixing-related quantities:

1. `ComputeAdjustedTurbulentViscosity`: Calculates turbulent viscosity (ν_T) adjusted by the gradient Richardson number to account for stratification effects.
2. `ComputeShearStrainProduction`: Computes TKE production due to shear strain, focusing on vertical shear components.
3. `ComputeAverageRichardsonNumber`: Calculates the average Richardson number across all cells for diagnostic output.

Turbulent Viscosity Computation

The `ComputeAdjustedTurbulentViscosity` method computes turbulent viscosity for a given cell and sigma layer k:

1. **Base Turbulent Viscosity**:
$$\nu_T = c_\mu \frac{k^2}{\epsilon} \tag{1}$$
where k is TKE (m^2/s^2) and ϵ is the dissipation rate (m^2/s^3). If $\epsilon < 10^{-8}$ or $k < 10^{-6}$, or if ν_T is NaN/infinite, $\nu_T = \nu = 10^{-6}\,\text{m}^2/\text{s}$.
2. **Buoyancy Frequency Squared** (N^2):
$$N^2 = -\frac{g}{\rho_0}\frac{\partial \rho}{\partial z}, \quad \frac{\partial \rho}{\partial z} = \frac{\rho_{k+1} - \rho_k}{\Delta\sigma \cdot h} \tag{2}$$
where $\rho_k = \rho_0 + 0.8 S_k$, S_k is salinity (PSU), h is depth (m), and $\Delta\sigma = 1/n_\sigma$. $N^2 \in [0, 10^{-3}]\,\text{s}^{-2}$.

3. **Shear Squared** (S^2):
$$S^2 = \text{Shear}_k^2, \quad S^2 \in [10^{-6}, 100]\,\text{s}^{-2} \tag{3}$$
where Shear_k is the velocity gradient (s^{-1}).

4. **Gradient Richardson Number**:
$$Ri = \frac{N^2}{S^2}, \quad Ri \in [0, 10] \tag{4}$$

5. **Stability Function**:
$$f(Ri) = \frac{1}{1 + 10Ri}, \quad f(Ri) \in [0.1, 1.0] \tag{5}$$
This reduces mixing when $Ri > 0.25$, indicating stable stratification.

6. **Adjusted Turbulent Viscosity**:
$$\nu_T \leftarrow \nu_T \cdot f(Ri), \quad \nu_T \in [10^{-6}, 10^{-2}]\,\text{m}^2/\text{s} \tag{6}$$

Shear-Strain-Induced TKE Production

The `ComputeShearStrainProduction` method calculates TKE production for layer k:

1. **Vertical Shear Components**:
$$\frac{\partial u}{\partial z} = \frac{u_k - u_{k+1}}{\Delta\sigma \cdot h}, \quad \frac{\partial u}{\partial z} \in [-100, 100]\,\text{s}^{-1} \tag{7}$$
$$\frac{\partial v}{\partial z} = \frac{v_k - v_{k+1}}{\Delta\sigma \cdot h}, \quad \frac{\partial v}{\partial z} \in [-100, 100]\,\text{s}^{-1} \tag{8}$$
where u_k, v_k are horizontal velocities (m/s).

2. **Shear-Strain Production**:
$$P = \nu_T \cdot \left(\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right), \quad P \in [0, 10^{-3}]\,\text{m}^2/\text{s}^3 \tag{9}$$
No production is computed at the top layer ($k = n_\sigma - 1$).

Average Richardson Number

The `ComputeAverageRichardsonNumber` method computes the average gradient Richardson number across all cells and layers:

1. For each cell and layer $k < n_\sigma - 1$:
$$N^2 = -\frac{g}{\rho_0} \frac{\rho_{k+1} - \rho_k}{\Delta\sigma \cdot h}, \quad N^2 \in [0, 10^{-3}]\,\text{s}^{-2} \tag{10}$$
$$S^2 = \text{Shear}_k^2, \quad S^2 \in [10^{-6}, 100]\,\text{s}^{-2} \tag{11}$$
$$Ri = \frac{N^2}{S^2}, \quad Ri \in [0, 10] \tag{12}$$

2. Sum valid Ri values (excluding NaN/infinite) and average over valid cells/layers.

Physical and Mathematical Models

The `RichardsonNumDepAndSSIMix` class employs the following models:

- **Turbulent Viscosity**:

$$\nu_T = c_\mu \frac{k^2}{\epsilon} \cdot \frac{1}{1 + 10 \cdot \frac{N^2}{S^2}}, \quad \nu_T \in [10^{-6}, 10^{-2}] \tag{13}$$

$$N^2 = -\frac{g}{\rho_0} \frac{\rho_{k+1} - \rho_k}{\Delta\sigma \cdot h}, \quad N^2 \in [0, 10^{-3}] \tag{14}$$

$$S^2 = \text{Shear}_k^2, \quad S^2 \in [10^{-6}, 100] \tag{15}$$

- **Shear-Strain TKE Production**:

$$P = \nu_T \cdot \left(\left(\frac{u_k - u_{k+1}}{\Delta\sigma \cdot h} \right)^2 + \left(\frac{v_k - v_{k+1}}{\Delta\sigma \cdot h} \right)^2 \right), \quad P \in [0, 10^{-3}] \tag{16}$$

- **Average Richardson Number**:

$$Ri_{\text{avg}} = \frac{1}{N} \sum_{\text{valid}} \frac{N^2}{S^2}, \quad Ri \in [0, 10] \tag{17}$$

These models capture stratification effects via the Richardson number and shear-driven turbulence, enhancing the accuracy of mixing processes in estuarine simulations.

Lattice Boltzmann and Finite Volume Large Eddy Simulations

This document describes two C# classes, `LatticeBoltzmannLES` and `LargeEddySim`, designed for large eddy simulation (LES) of estuarine circulation, incorporating river inflow, tidal forcing, and salinity-driven density variations. Both classes use the Smagorinsky model for subgrid-scale turbulence and provide graphical user interfaces (GUIs) for simulation control and visualization.

LatticeBoltzmannLES

The `LatticeBoltzmannLES` class implements a 2D lattice Boltzmann method (LBM) with the D2Q9 model to simulate estuarine hydrodynamics, capturing velocity, salinity, and turbulence quantities.

Core Components and Initialization

The class is initialized with:

- Grid size: $n_x = 200$, $n_y = 50$ (lattice points)
- Spatial step: $\Delta x = 50\,\text{m}$
- Time step: $\Delta t = 0.1\,\text{s}$ (reserved for future scaling)
- River inflow: $u_{\text{river}} = 0.1\,\text{m/s}$
- Tidal amplitude: $A_{\text{tidal}} = 1.0\,\text{m}$, period: $T_{\text{tidal}} = 43200\,\text{s}$
- Smagorinsky constant: $C_s = 0.1$
- Reynolds number: $Re = 1000$
- Salinity: $S_{\text{ocean}} = 35\,\text{PSU}$, $S_{\text{river}} = 0\,\text{PSU}$
- Ocean temperature: $T_{\text{ocean}} = 20\,°\text{C}$

Arrays include:

- Density: $\rho(i, j)$
- Salinity: $S(i, j)$
- Velocity distribution functions: $f(i, j, k)$, equilibrium $f^{\text{eq}}(i, j, k)$ (D2Q9, $k = 0, \ldots, 8$)
- Salinity distribution functions: $g(i, j, k)$, equilibrium $g^{\text{eq}}(i, j, k)$
- Velocities: $u(i, j)$, $v(i, j)$; time-averaged: $u_{\text{avg}}(i, j)$, $v_{\text{avg}}(i, j)$
- Vorticity or Q-criterion: $\omega(i, j)$
- Local Smagorinsky constant: $C_s(i, j)$
- Local spatial step: $\Delta x_{\text{local}}(i, j)$, relaxation times: $\tau(i, j)$, $\tau_S(i, j)$

The GUI allows users to adjust parameters and select visualization modes (velocity, salinity, time-averaged velocity), refinement modes (None, River, Tidal, Both), filter width (Lattice Spacing, Cell Area), and vortex visualization (Vorticity Magnitude, Q-Criterion).

Functioning Logic

The simulation follows the LBM algorithm:

1. **Streaming**: Propagate $f(i,j,k)$ and $g(i,j,k)$ along D2Q9 directions:
$$f(i+c_{k,x}, j+c_{k,y}, k) = f(i,j,k), \quad g(i+c_{k,x}, j+c_{k,y}, k) = g(i,j,k) \tag{1}$$
where $c_k = (c_{k,x}, c_{k,y})$ are lattice directions.

2. **Collision**: Update distributions using the Bhatnagar-Gross-Krook (BGK) model:
$$f(i,j,k) \leftarrow f(i,j,k) + \frac{f^{\text{eq}}(i,j,k) - f(i,j,k)}{\tau(i,j)} \tag{2}$$
$$g(i,j,k) \leftarrow g(i,j,k) + \frac{g^{\text{eq}}(i,j,k) - g(i,j,k)}{\tau_S(i,j)} \tag{3}$$
where equilibrium distributions are:
$$f^{\text{eq}}(i,j,k) = w_k \rho \left[1 + 3\frac{\mathbf{u}\cdot\mathbf{c}_k\Delta x}{\Delta x_{\text{local}}} + 4.5\left(\frac{\mathbf{u}\cdot\mathbf{c}_k\Delta x}{\Delta x_{\text{local}}}\right)^2 - 1.5\frac{|\mathbf{u}|^2\Delta x^2}{\Delta x_{\text{local}}^2}\right] \tag{4}$$
$$g^{\text{eq}}(i,j,k) = w_k S \rho \left[1 + 3\frac{\mathbf{u}\cdot\mathbf{c}_k\Delta x}{\Delta x_{\text{local}}} + 4.5\left(\frac{\mathbf{u}\cdot\mathbf{c}_k\Delta x}{\Delta x_{\text{local}}}\right)^2 - 1.5\frac{|\mathbf{u}|^2\Delta x^2}{\Delta x_{\text{local}}^2}\right] \tag{5}$$
with weights $w_k = \{4/9, 1/9, \ldots, 1/36\}$.

3. **Macroscopic Variables**: Compute density, velocity, and salinity:
$$\rho(i,j) = \sum_{k=0}^{8} f(i,j,k), \quad u(i,j) = \frac{1}{\rho}\sum_{k=0}^{8} f(i,j,k)c_{k,x}, \quad v(i,j) = \frac{1}{\rho}\sum_{k=0}^{8} f(i,j,k)c_{k,y}, \quad S(i,j) = \frac{1}{\rho}\sum_{k=} \tag{6}$$

4. **Turbulence Modeling**: Use Smagorinsky LES:
$$\nu_t = (C_s\Delta)^2|\mathbf{S}|, \quad \tau(i,j) = 3(\nu+\nu_t)c_s^2\frac{\Delta x}{\Delta x_{\text{local}}} + 0.5 \tag{7}$$
where Δ is the filter width (Δx_{local} or $\sqrt{\Delta x_{\text{local}}^2}$), $|\mathbf{S}|$ is the strain rate magnitude, and $c_s^2 = 1/3$.

5. **Vorticity/Q-Criterion**: Compute vorticity ($|\partial v/\partial x - \partial u/\partial y|$) or Q-criterion:
$$Q = \frac{1}{2}\left(\|\Omega\|^2 - \|\mathbf{S}\|^2\right), \quad \Omega_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right) \tag{8}$$

6. **Boundary Conditions**: Apply river inflow ($u = u_{\text{river}}, S = S_{\text{river}}$) at $x = 0$ and tidal velocity ($u = A_{\text{tidal}}\sin(2\pi t/T_{\text{tidal}})/\Delta x, S = S_{\text{ocean}}$) at $x = n_x - 1$.

7. **Energy Spectrum**: Compute FFT of velocities at mid-y plane every 100 timesteps, outputting power spectral density to `energy_spectrum.txt`.

Visualization

The GUI displays:

- Velocity, salinity, or time-averaged velocity in a color-coded panel (red-blue gradient).
- Vorticity or Q-criterion isosurfaces in a separate panel, colored by velocity or vorticity.
- Console output with time, average velocity, salinity, TKE, dissipation, and vorticity.

LargeEddySim

The `LargeEddySim` class implements a 2D finite volume method for LES, solving Navier-Stokes equations with salinity and temperature effects.

Core Components and Initialization

The class is initialized with:

- Grid size: $n_x = 50$, $n_z = 20$
- Estuary dimensions: $L = 10000\,\text{m}$, $h = 10\,\text{m}$
- Spatial steps: $\Delta x = L/n_x$, $\Delta z = h/n_z$
- Time step: $\Delta t = 1.0\,\text{s}$ (adjusted by CFL condition)
- Smagorinsky constant: $C_s = 0.1$
- Kinematic viscosity: $\nu = 10^{-6}\,\text{m}^2/\text{s}$
- River inflow: $Q_{\text{river}} = 0.1\,\text{m}^3/\text{s}$
- Tidal amplitude: $A_{\text{tidal}} = 1.0\,\text{m}$, period: $T_{\text{tidal}} = 43200\,\text{s}$
- Ocean salinity: $S_{\text{ocean}} = 35\,\text{PSU}$
- Ocean temperature: $T_{\text{ocean}} = 20\,°\text{C}$

Arrays include:

- Velocities: $u(i,j)$, $w(i,j)$
- Salinity: $S(i,j)$, temperature: $T(i,j)$
- Vorticity: $\omega(i,j)$
- Eddy viscosity: $\nu_e(i,j)$

The GUI allows parameter adjustments and selection of time integration schemes (RK4 or Crank-Nicolson).

Functioning Logic

The simulation follows:

1. **Density and Pressure Gradient**: Compute density using the equation of state:

$$\rho(i,j) = \text{EqOfState}(S(i,j), T(i,j), P), \quad P = \rho_0 g(h - j\Delta z)/10^4 \tag{9}$$

   Pressure gradient: $\partial P/\partial x = g(\rho_{i+1,j} - \rho_{i-1,j})/(2\Delta x)$.

2. **Eddy Viscosity**: Use Smagorinsky model:

$$\nu_e = (C_s\Delta)^2|\mathbf{S}|, \quad \Delta = \sqrt{\Delta x \Delta z}, \quad |\mathbf{S}| = \sqrt{2\left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial z}\right)^2\right) + \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)^2} \tag{10}$$

3. **Tendencies**: Compute time derivatives for velocity, salinity, and temperature:

$$\frac{\partial u}{\partial t} = -\left(u\frac{\partial u}{\partial x} + w\frac{\partial u}{\partial z}\right) + \frac{1}{\rho_0}\frac{\partial}{\partial x}\left(\nu_e\frac{\partial u}{\partial x}\right) + \frac{1}{\rho_0}\frac{\partial}{\partial z}\left(\nu_e\frac{\partial u}{\partial z}\right) - \frac{1}{\rho_0}\frac{\partial P}{\partial x} \tag{11}$$

$$\frac{\partial w}{\partial t} = -\left(u\frac{\partial w}{\partial x} + w\frac{\partial w}{\partial z}\right) + \frac{1}{\rho_0}\frac{\partial}{\partial x}\left(\nu_e\frac{\partial w}{\partial x}\right) + \frac{1}{\rho_0}\frac{\partial}{\partial z}\left(\nu_e\frac{\partial w}{\partial z}\right) \tag{12}$$

$$\frac{\partial S}{\partial t} = -\left(u\frac{\partial S}{\partial x} + w\frac{\partial S}{\partial z}\right) + \frac{\partial}{\partial x}\left(\nu_e\frac{\partial S}{\partial x}\right) + \frac{\partial}{\partial z}\left(\nu_e\frac{\partial S}{\partial z}\right) \tag{13}$$

$$\frac{\partial T}{\partial t} = -\left(u\frac{\partial T}{\partial x} + w\frac{\partial T}{\partial z}\right) + \frac{\partial}{\partial x}\left(\nu_e\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial z}\left(\nu_e\frac{\partial T}{\partial z}\right) \tag{14}$$

4. **Time Integration**: Use RK4 or Crank-Nicolson (CN):
   - **RK4**: Four-stage explicit method:

$$\phi^{n+1} = \phi^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{15}$$

   - **CN**: Implicit for diffusion, explicit for advection, solved using tridiagonal matrix algorithm.

5. **Boundary Conditions**: River inflow ($u = Q_{\text{river}}/h, S = 0$) at $x = 0$, tidal velocity ($u = A_{\text{tidal}}\cos(2\pi t/T_{\text{tidal}})$) at $x = L$, no-slip at bottom, and free surface.

6. **Vorticity**: Compute as:

$$\omega = \frac{\partial w}{\partial x} - \frac{\partial u}{\partial z} \tag{16}$$

Visualization

The GUI displays:

- Vorticity field (blue-red gradient).
- Velocity vectors (black arrows).
- Status with simulation time, scheme, and Δt.

Comparison and Physical Models

Both classes model estuarine circulation with:

- **Navier-Stokes Equations**: Solved via LBM (D2Q9) in `LatticeBoltzmannLES` or finite volume in `LargeEddySim`.
- **Smagorinsky Model**: Eddy viscosity:

$$\nu_e = (C_s \Delta)^2 |\mathbf{S}| \tag{17}$$

- **Density**: $\rho = 1000(1 + 0.0008S - 0.00007(T - 20))$ in `LatticeBoltzmannLES`; custom `EqOfState` in `LargeEddySim`.
- **Boundary Conditions**: River inflow and tidal forcing drive the flow.

`LatticeBoltzmannLES` uses a lattice-based approach with adaptive grid refinement and energy spectrum output, while `LargeEddySim` uses a finite volume method with RK4 or Crank-Nicolson integration, suitable for structured grids.

Spectral Analysis Tool

The `SpectralAnalyzer` class, implemented in C# within the `EstuarineCirculationModeli` namespace, provides a graphical user interface (GUI) for spectral and Empirical Orthogonal Function (EOF)/Proper Orthogonal Decomposition (POD) analysis of estuarine circulation data. It processes time series (e.g., Richardson number, velocity, water level) and spatial-temporal data (e.g., salinity, velocity profiles) to analyze tidal dynamics, turbulence, and mixing events.

Simulation Logic

The class simulates data collection and analysis through a timer-driven loop (`UpdateSimulation`), with a default interval of 100 ms. The simulation progresses until the specified duration ($T = 86400$ s, one day) is reached, collecting data at a user-defined sampling rate ($\Delta t = 3600$ s, one sample per hour). The GUI allows users to:

- Select analysis type: Spectral Analysis or EOF/POD Analysis.
- Choose variables: Richardson number, velocity, water level (for spectral analysis); salinity, velocity profile (for EOF/POD).
- Set plot scale (linear or log-log), window type (Hanning, Hamming, Blackman, Rectangular), and EOF/POD mode (1, 2, or 3).
- Control simulation via Analyze, Pause, Reset, and Clear buttons.

Data is stored in:

- `timeSeriesData`: List of scalar values for spectral analysis.
- `salinityData`: List of 2D arrays $[100, 1]$ for salinity (10x10 grid flattened).
- `velocityData`: List of 2D arrays $[10, 1]$ for velocity profiles (10 depth levels).
- `mixingEvents`: List of detected mixing events with time and value.

The simulation logic:

1. **Initialization**: Clears data, sets `currentTime = 0`, and starts the timer (`isAnalyzing = true`).
2. **Data Generation**: At each timestep ($t \leftarrow t + \Delta t$):
    - For spectral analysis, generates synthetic data for the selected variable (e.g., Richardson number) using `GenerateSyntheticData`.
    - For EOF/POD, generates salinity (`GenerateSyntheticSalinityData`) or velocity (`GenerateSyntheticVelocityData`).
    - Detects mixing events by computing the standard deviation over a window (size 5) and comparing against thresholds (0.2 for Richardson number, 0.03 for velocity, 0.5 for water level).

3. **Analysis**: When `currentTime` reaches `duration`, the timer stops, and `PerformAnalysi` triggers either `PerformSpectralAnalysis` or `PerformEOFAnalysis`.
4. **Visualization**: Updates two panels:
    - `visualizationPanel`: Plots power spectral density (PSD) or time series (spectral analysis), or spatial modes (EOF/POD).
    - `heatmapPanel`: Plots time-frequency heatmap (spectral analysis) or temporal coefficients (EOF/POD).

Physical and Mathematical Models

Synthetic Data Generation

Synthetic data mimics estuarine dynamics with tidal and inertial influences:

- **Richardson Number**: Models stratification stability:

$$Ri(t) = 0.5 + 0.3 \sin\left(\frac{2\pi t}{T_{\text{tidal}}}\right) + 0.2 \sin\left(\frac{2\pi t}{T_{\text{inertial}}}\right) + \text{noise} \tag{1}$$

  where $T_{\text{tidal}} = 43200\,\text{s}$ (12-hour semi-diurnal tide), $T_{\text{inertial}} = 17 \times 3600\,\text{s}$ (inertial period at 45° latitude), and noise is a random jump (~ 0.5 with 2% probability).

- **Velocity**: Represents tidal-driven flow:

$$u(t) = 0.1 + 0.05 \sin\left(\frac{2\pi t}{T_{\text{tidal}}}\right) + 0.02 \sin\left(\frac{2\pi t}{T_{\text{tidal}}/2}\right) + \text{noise} \tag{2}$$

  where noise is random (~ 0.01).

- **Water Level**: Combines tidal constituents:

$$\eta(t) = 1.0 \sin\left(\frac{2\pi t}{T_{\text{M2}}}\right) + 0.5 \sin\left(\frac{2\pi t}{T_{\text{K1}}}\right) \tag{3}$$

  where $T_{\text{M2}} = 44712\,\text{s}$ (M2 tide), $T_{\text{K1}} = 86148\,\text{s}$ (K1 tide).

- **Salinity**: Models spatial-temporal variations:

$$S(x, y, t) = \left[30.0 + 2.0 \sin\left(\frac{2\pi t}{T_{\text{M2}}}\right)\right](1.0 - 0.1(x + y)) + \text{noise} \tag{4}$$

  on a 10x10 grid, with noise (~ 0.5).

- **Velocity Profile**: Depth-dependent tidal flow:

$$u(z, t) = \left[0.1 \sin\left(\frac{2\pi t}{T_{\text{M2}}}\right)\right](1.0 - 0.1z) + \text{noise} \tag{5}$$

  for 10 depth levels, with noise (~ 0.01).

Spectral Analysis

Spectral analysis uses Welch's method to estimate the power spectral density (PSD):

1. **Segmentation**: Divides the time series into overlapping segments (length $N = 2^{\lfloor \log_2 \min(256, |\text{data}|) \rfloor}$, overlap $N/2$).
2. **Windowing**: Applies a window function:
   - Hanning: $w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$
   - Hamming: $w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$
   - Blackman: $w(n) = 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right)$
   - Rectangular: $w(n) = 1.0$

   Window power: $P_w = \frac{1}{N}\sum_{n=0}^{N-1} w(n)^2$.
3. **FFT**: Computes the Fast Fourier Transform (FFT) for each segment using the Cooley-Tukey algorithm:

$$X(k) = \sum_{n=0}^{N-1} x(n)w(n)e^{-i\frac{2\pi kn}{N}}, \quad k = 0, \ldots, \frac{N}{2} \tag{6}$$

4. **PSD**: Calculates power spectrum:

$$\text{PSD}(k) = \frac{1}{N_s P_w \Delta t} \sum_{\text{segments}} |X(k)|^2 \tag{7}$$

   where N_s is the number of segments, and frequencies are $f_k = \frac{k}{N\Delta t}$.
5. **Short-Time Fourier Transform (STFT)**: Computes time-frequency heatmap:

$$\text{STFT}(k, s) = \left| \sum_{n=0}^{N-1} x(n + s(N - \text{overlap}))w(n)e^{-i\frac{2\pi kn}{N}} \right|^2 \frac{1}{P_w \Delta t} \tag{8}$$

   for segment s.
6. **Turbulence Detection**: Checks for a $-5/3$ spectral slope in the log-log PSD (first 10 frequencies, $f > 10^{-4}$ Hz):

$$\text{slope} = \frac{\sum(\log f_i - \overline{\log} f)(\log \text{PSD}_i - \overline{\log} \text{PSD})}{\sum(\log f_i - \overline{\log} f)^2} \tag{9}$$

   Turbulence is detected if $|\text{slope} + \frac{5}{3}| < 0.5$.
7. **Tidal Constituents**: Identifies M2 ($f_{\text{M2}} = \frac{1}{44712}$ Hz) and K1 ($f_{\text{K1}} = \frac{1}{86148}$ Hz) frequencies within $2\Delta f = \frac{2}{N\Delta t}$.

EOF/POD Analysis

EOF/POD decomposes spatial-temporal data into orthogonal modes:

1. **Data Matrix**: For salinity ($100 \times T$) or velocity ($10 \times T$), where T is the number of time steps. Mean subtraction per spatial point:

$$X(s,t) \leftarrow X(s,t) - \frac{1}{T}\sum_{t=1}^{T} X(s,t) \tag{10}$$

2. **Singular Value Decomposition (SVD)**: Approximated via power iteration for the top 3 modes:
   - Initialize random vector $\mathbf{v}$, normalize: $\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}$.
   - Iterate (20 times):

$$\mathbf{u} = X\mathbf{v}, \quad \mathbf{u} \leftarrow \frac{\mathbf{u}}{\|\mathbf{u}\|} \tag{11}$$

$$\mathbf{v} \leftarrow X^T\mathbf{u}, \quad \mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|} \tag{12}$$

   - Compute singular value: $\sigma = \|X\mathbf{v}\|$, spatial mode: $\mathbf{u} = \frac{X\mathbf{v}}{\sigma}$, temporal coefficients: $\mathbf{v}$.
   - Deflate: $X \leftarrow X - \sigma\mathbf{u}\mathbf{v}^T$.
3. **Explained Variance**: For singular values σ_i:

$$\text{Variance}_i = \frac{\sigma_i^2}{\sum \sigma_j^2} \times 100\% \tag{13}$$

Mixing Events

Mixing events are detected by computing the standard deviation over a window of 5 samples:

$$\sigma = \sqrt{\frac{1}{5}\sum_{i=1}^{5}(x_i - \bar{x})^2} \tag{14}$$

A mixing event is flagged if σ exceeds the threshold (0.2 for Richardson number, 0.03 for velocity, 0.5 for water level).

Visualization

- **Spectral Analysis**:
  - `visualizationPanel`: Plots PSD (linear or log-log) or time series with mixing events (red dots). M2 and K1 frequencies are marked.
  - `heatmapPanel`: Displays STFT heatmap, with power scaled as $\log_{10}(\text{STFT})$ (log-log) or linear, using red-blue color gradient.

- **EOF/POD Analysis**:
  - `visualizationPanel`: For salinity, plots spatial mode on a 10x10 grid (red-blue gradient); for velocity, plots mode vs. depth.
  - `heatmapPanel`: Plots temporal coefficients vs. time.

Adaptive Mesh Refinement for Estuarine Modeling

The `AdaptiveMeshRef` class, implemented in C# within the `EstuarineCirculationModelin` namespace, simulates estuarine circulation with adaptive mesh refinement (AMR) for salinity, temperature, sediment, and velocity fields. It dynamically refines a 2D grid (400×100) based on gradients in key variables, improving resolution in regions of high variability (e.g., estuarine turbidity maximum, fronts) while modeling tidal, wind, riverine, and Coriolis effects.

Simulation Logic

The class operates on a base grid ($N_x = 400$, $N_z = 100$) over an estuary of specified length and depth, with refinement applied to cells exhibiting high gradients. Key features include:

- **Initialization**: Stores input profiles for salinity, temperature, sediment, horizontal (u) and vertical (w) velocities, and bedload, along with physical parameters (e.g., viscosity, diffusivities, settling velocity, tidal and wind forcing).
- **Data Structures**:
  - Base grid arrays ($[400, 100]$) for salinity, temperature, sediment, and velocities.
  - `bedloadProfile` ($[400]$) for bottom sediment transport.
  - `refineFlags` ($[400, 100]$) to mark cells for 2×2 refinement.
  - Dictionaries (`fineSalinityGrids`, etc.) for refined 2×2 subgrids.
- **Refinement**: The `RefineGrid` method flags cells with gradient magnitudes in the top 10% (threshold factor 0.9) and initializes fine grids using bilinear interpolation.
- **Update**: The `UpdateFields` method advances fields using smaller timesteps ($\Delta t/4$) in refined regions, applying boundary conditions and updating coarse grid values by averaging fine grid results.

Simulation steps:

1. **Gradient-Based Refinement**: Computes gradients for salinity, temperature, and u-velocity, flagging cells where the combined gradient magnitude exceeds a threshold.
2. **Field Updates**: Updates refined regions with substeps, then updates coarse grid and bedload.
3. **Access Methods**: Provides `GetSalinity`, `GetTemperature`, `GetSediment`, `GetUVelocity`, and `GetWVelocity` to retrieve values from fine or coarse grids.

Physical and Mathematical Models

Grid Refinement

The base grid has spacings $\Delta x = L/(N_x - 1)$, $\Delta z = H/(N_z - 1)$, where L is estuary length and H is depth. Refined cells use $\Delta x_{\text{fine}} = \Delta x/2$, $\Delta z_{\text{fine}} = \Delta z/2$, and timestep $\Delta t_{\text{fine}} = \Delta t/4$. Gradient magnitude for cell (i, j) is:

$$G = \sqrt{\left(\frac{\partial S}{\partial x}\right)^2 + \left(\frac{\partial S}{\partial z}\right)^2 + \left(\frac{\partial T}{\partial x}\right)^2 + \left(\frac{\partial T}{\partial z}\right)^2 + \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2} \tag{1}$$

where $\frac{\partial S}{\partial x} \approx \frac{S_{i+1,j} - S_{i-1,j}}{2\Delta x}$, etc. Cells with G in the top 10% are refined.

Velocity Update

Horizontal velocity u in refined cells is updated using:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - w\frac{\partial u}{\partial z} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2}\right) - fw + \frac{u_{\text{tidal}} - u}{T_{\text{tidal}}} + u_{\text{wind}} + \xi \tag{2}$$

where ν is kinematic viscosity, f is the Coriolis parameter, $u_{\text{tidal}} = A\cos(2\pi t/T_{\text{tidal}})$, T_{tidal} is tidal period, A is tidal amplitude, $u_{\text{wind}} = \frac{\tau_{wx}}{\rho_0 \nu} e^{-z/\delta}$ for surface cells ($\delta = \sqrt{2\nu/|f|}$), and $\xi = (k_w \tau_w (1 - z/H) + I)A(r - 0.5)$ adds wind mixing and turbulence ($r \sim \text{Uniform}[0, 1]$). Vertical velocity w is updated similarly:

$$\frac{\partial w}{\partial t} = -u\frac{\partial w}{\partial x} - w\frac{\partial w}{\partial z} + \nu\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial z^2}\right) + fu + w_{\text{wind}} + \xi \tag{3}$$

Boundary conditions:

- River ($x = 0$): $u = Q(t)$, $w = 0$.
- Ocean ($x = L$): $u = u_{\text{tidal}}$, $w = 0$.
- Bottom ($z = 0$): $u = w = 0$.
- Surface ($z = H$): $u = u_{\text{tidal}} + \tau_{wx}/(\rho_0 \nu)$, $w = \tau_{wz}/(\rho_0 \nu)$.

Wind stress is $\tau_w = \rho_a C_d U_w^2$, with $U_w = U_{\text{wind}}|\sin(2\pi t/T_{\text{wind}})|$, $\rho_a = 1.225\,\text{kg/m}^3$, $C_d = 0.0012$.

Salinity and Temperature Update

Salinity S is updated using:

$$\frac{\partial S}{\partial t} = -u\frac{\partial S}{\partial x} - w\frac{\partial S}{\partial z} + \nu\left(\frac{\partial^2 S}{\partial x^2} + \frac{\partial^2 S}{\partial z^2}\right) \tag{4}$$

Temperature T uses thermal diffusivity κ:

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - w\frac{\partial T}{\partial z} + \kappa\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2}\right) \tag{5}$$

Boundary conditions:

- River: $S = 10(1 - Q(t)/Q_{\text{max}})$, $T = 14 - 4Q(t)/Q_{\text{max}}$.
- Ocean: $S = 35$, $T = 15$.
- Bottom: $\partial S/\partial z = 0$, $\partial T/\partial z = 0$.
- Surface: $\partial S/\partial z = 0$, $T = 15$.

River discharge is $Q(t) = 0.1 + A_r \sin(2\pi t/T_r)$, A_r is river discharge amplitude, T_r is river period.

Sediment Transport

Suspended sediment concentration C is updated with settling:

$$\frac{\partial C}{\partial t} = -u\frac{\partial C}{\partial x} - (w + w_s)\frac{\partial C}{\partial z} + D_s\left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial z^2}\right) \tag{6}$$

where w_s is settling velocity, D_s is sediment diffusivity, and $C \geq 0$. Boundary conditions:

- River: $C = 200Q(t)/Q_{\text{max}}$.
- Ocean: $C = 0$.
- Bottom: $\partial C/\partial z = 0$.
- Surface: $\partial C/\partial z = 0$.

Bedload Transport

Bedload q_b at $z = 0$ is updated using:

$$\frac{\partial q_b}{\partial t} = E - D - \frac{\partial q_b}{\partial x} \tag{7}$$

where erosion rate $E = \alpha_e u_*^2$, $u_* = \sqrt{|\tau_b|/\rho_0}$, $\tau_b = \rho_0 \nu(\partial u/\partial z)|_{z=0}$, deposition rate $D = -w_s C(x, z = 1)$, and bedload flux:

$$q_b = \begin{cases} 8\left(\theta - \theta_c\right)^{1.5}\sqrt{\left(\frac{\rho_s}{\rho_0} - 1\right)gd^3}, & \theta > \theta_c \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where $\theta = \tau_b/((\rho_s - \rho_0)gd)$, $\theta_c = 0.047$, $\rho_s = 2650\,\text{kg/m}^3$, $d = 0.001\,\text{m}$, $\alpha_e = 0.0001$.

Notes

- The model uses central differencing for spatial derivatives and explicit time integration.
- Fine grids are updated with 2×2 substeps to maintain stability (CFL condition).

- Turbulence and wind mixing introduce stochastic perturbations proportional to tidal amplitude.
- No GUI is implemented, but fields can be accessed for external visualization.