A Generic “Gridding Engine” for 2D Modeling of Earth-Surface Dynamics

Gregory E. Tucker[1], Nicole Gasparini, Erkan Istanbulluoglu[2], Eric Hutton[3], Dan Hobley[1]

[1] CIRES and Department of Geological Sciences, University of Colorado, Boulder
[2] Department of Civil and Environmental Engineering, University of Washington, Seattle
[3] Community Surface Dynamics Modeling System (CSDMS), University of Colorado, Boulder

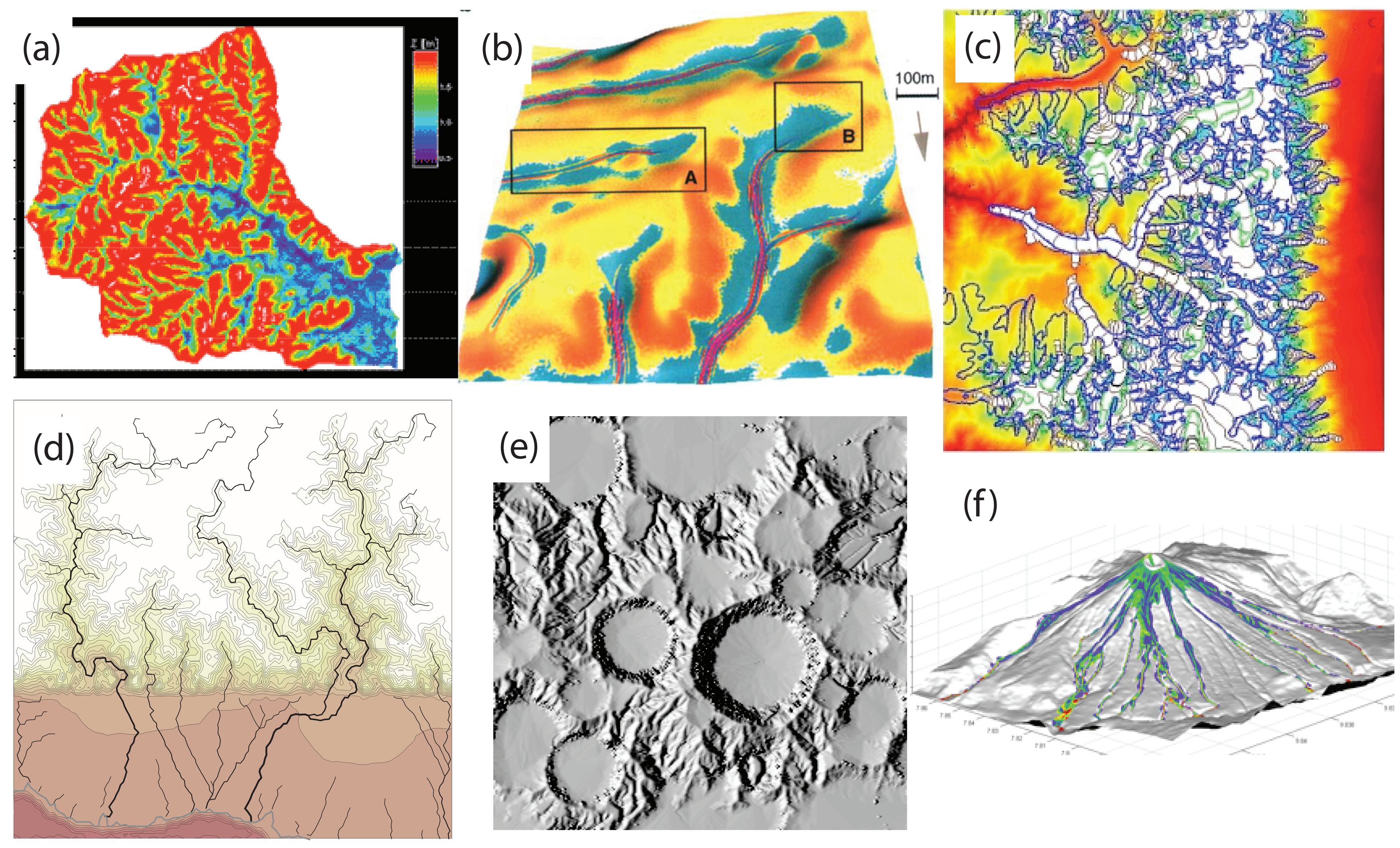
Abstract

This project addresses an important limitation to scientific productivity in fields that rely on computational modeling of landscape processes. Landscape models compute flows of mass, such as water, sediment, glacial ice, volcanic material, or landslide debris, across a gridded terrain surface. Science and engineering applications of these models range from short-term flood forecasting to long-term landform evolution. At present, software development behind these models is highly compartmentalized and idiosyncratic, despite the strong similarity in core algorithms and data structures between otherwise diverse models.
We believe that progress across the range of fields that use landscape models can be transformed by introducing a component-based approach to software development. We are engaged in a proof-of-concept study in which an existing landscape model code is adapted and enhanced to provide a set of independent, interoperable components (written initially in C++). These include: (1) a gridding engine to handle both regular and unstructured meshes, (2) an interface for space-time rainfall input, (3) a surface hydrology component, (4) an erosion-deposition component, (5) a vegetation component and (6) a simulation driver. The components can communicate with each other in one of two ways: using a simple C++ driver script, or using the Community Surface Dynamics Modeling System (CSDMS) Model Coupling Framework.

A central element is the gridding engine, which provides the ability to rapidly instantiate and configure a 2D simulation grid. Initially, the grid is an unstructured Delaunay/Voronoi mesh. Because the internal representation of geometry and topology is quite generic-consisting of nodes (cells), directed edges, polygon faces, etc.—the software can be enhanced to provide other grid formats, such as a simple raster or a quad-tree representation. The gridding engine also provides basic capabilities for finite-volume numerics, such as calculation of scalar gradients between pairs of neighboring cells, and calculation of flux divergence within cells.

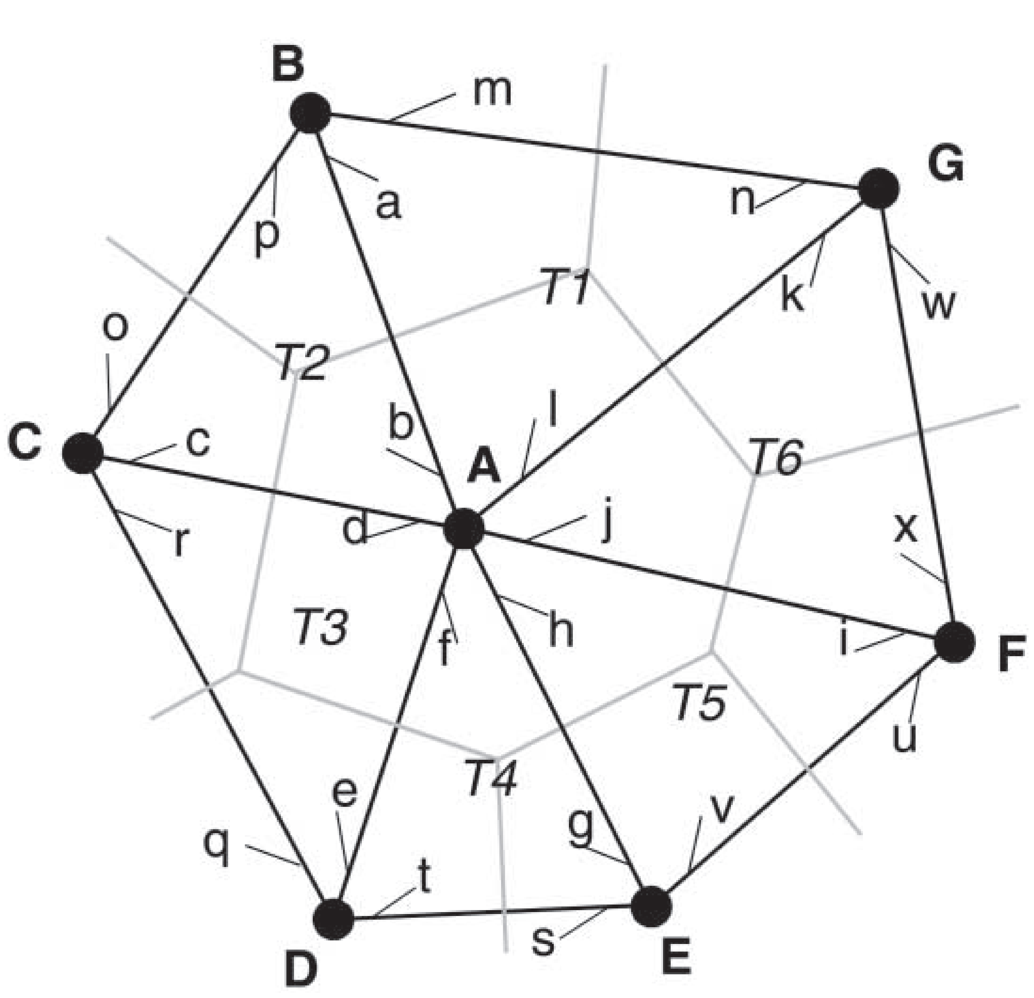
As they are developed, the components will be deployed to the surface-process community, and feedback will be collected from users, by (1) adding the components to the CSDMS library, and (2) providing hands-on training sessions at annual meetings of the CSDMS Terrestrial Working Group.

Our hope is that these interoperable and interchangeable components with simple, standardized interfaces, will transform the nature and speed of progress in the landscape sciences by allowing scientist-programmers to focus on the processes of interest rather than on the underlying software infrastructure.



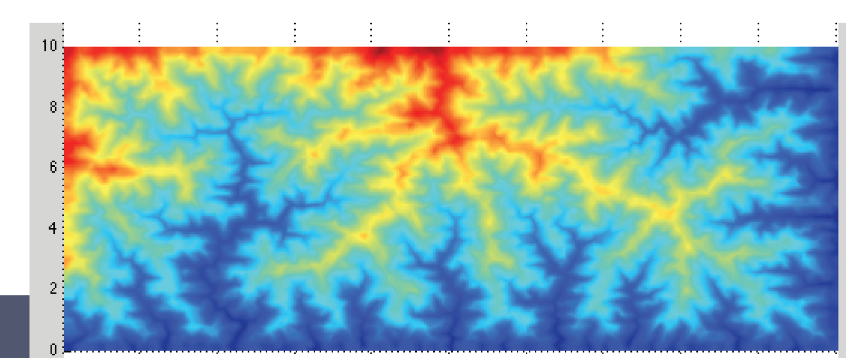
Examples of different types of landscape model. (a) Computed depth-to-groundwater, from the tRibs flood forecasting model (Ivanov et al., 2004). (b) Computed patterns of soil erosion and sedimentation on agricultural fields, using the SIMWE soil erosion model (Mitas and Mitasova, 1998). (c) Model of ice-age glacier extent over the Sierra Nevada Mountains, using the GC2D iceflow model (Kessler et al., 2006). (d) Simulation of canyon erosion and fan-delta progradation in a region of active uplift (top) and subsidence (bottom), using the CHILD landscape evolution model (Tucker and Hancock, 2010). (e) Model of simultaneous cratering and fluvial erosion on the ancient Mars surface, with the MARSSIM model (Howard, 2007). (f) Simulation of pyroclastic flows at Tungurahua volcano, Ecuador, using the VolcFlow model (Kelfoun et al., 2009).

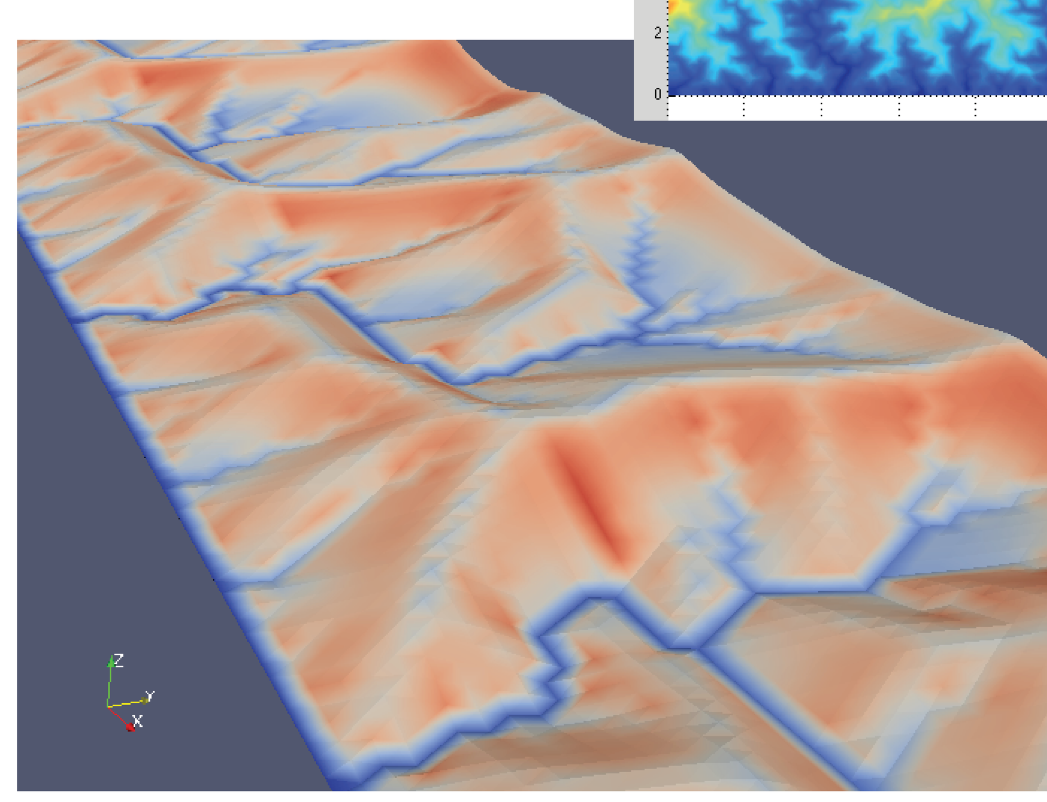
VORONOI / DELAUNAY MESH GEOMETRY



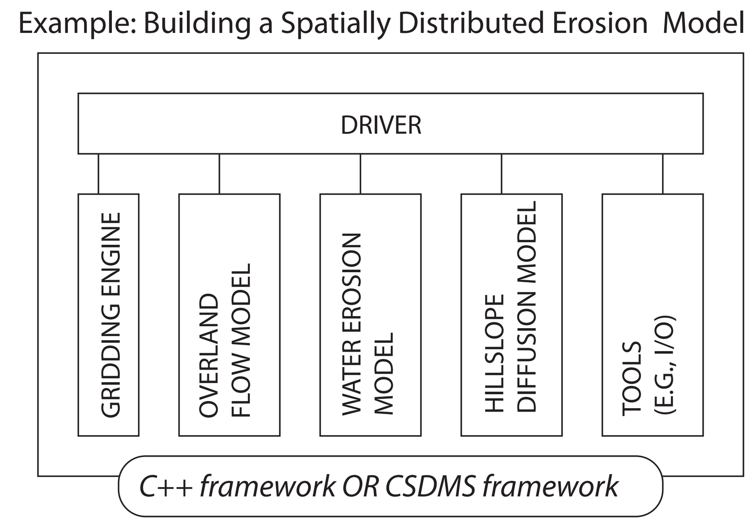
Sample mesh consisting of 7 nodes (A-G), 6 triangles (T1-T6), and 24 directed edges (a-x) (from Tucker et al., 2001).

CHILD model: source of code for gridding engine and other basic components

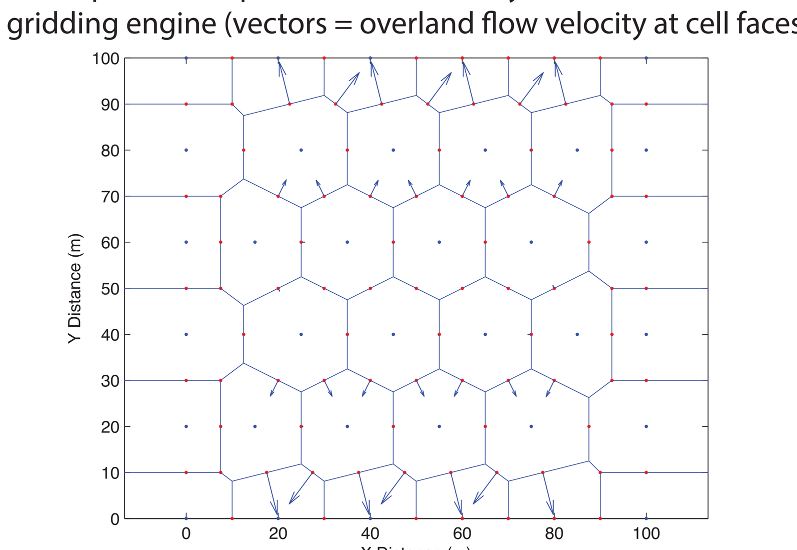




Example: Building a Spatially Distributed Erosion Model



Example of a simple Voronoi/Delaunay mesh created with gridding engine (vectors = overland flow velocity at cell faces)



EXAMPLE 1: TOPOGRAPHIC RIDGE

Tectonic uplift balances “diffusive” erosion

Driver

```
int main( int argc, char ** argv )
{
  std::cout << "Diffmod2de\n";

  Diffmod2de diffusion_model;
  std::string input_file_name;

  // Set the name of the input file
  std::cout << " setting input file name\n";
  if( argc>1 )
    input_file_name = argv[1];
  else
    input_file_name = "testdiffmod.in";

  // Open the input file
  std::cout << " opening input file\n";
  tInputFile infile( input_file_name.c_str() );

  // Find out how often, and in what format, to write output
  std::cout << " reading vars\n";
  double write_interval = infile.ReadDouble( "OPINTRVL" );
  double next_write = write_interval;

  // Read the run duration from the input file
  double runtime = infile.ReadDouble( "RUNTIME" );

  // Initialize the model
  std::cout << " about to initialize\n";
  diffusion_model.initialize( input_file_name );

  // Run the model
  double current_time = 0.0;
  while ( current_time < runtime )
  {
    double run_to_time = ( next_write < runtime ) ? next_writ
    diffusion_model.update_until( run_to_time );
    current_time = run_to_time;
    next_write += write_interval;
    diffusion_model.WriteOutputChildFormat( current_time );
  }

  // Clean up
  diffusion_model.finalize();

  std::cout << "Done with main\n" << std::flush;
}
```

Update function

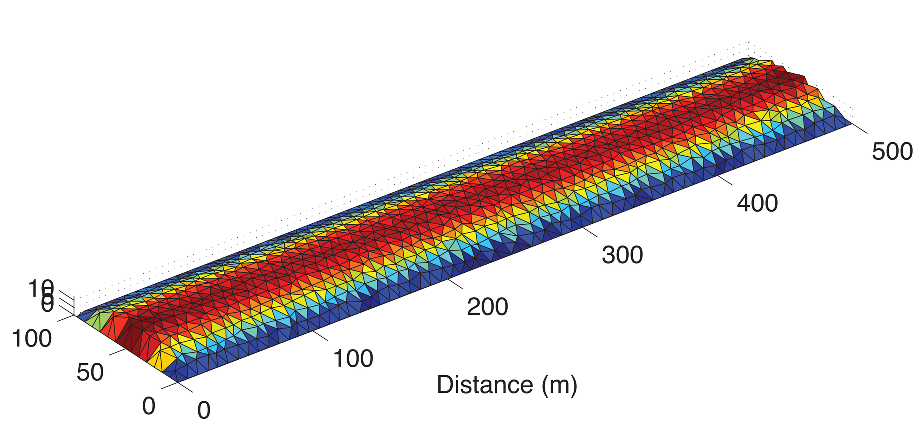
```
void Diffmod2de::
update()
{
  std::cout << "Diffmod2de::update()\n";

  // Calculate gradients along all edges
  mesh_->CalculateGradientsAlongEdges( eta_, S_ );

  // Calculate fluxes along edges
  for( unsigned j=0; j<nedges_; j++ )
    qs_[j] = -kd_ * S_[j];

  // Calculate flux divergence
  mesh_->CalculateFluxDivergenceAtCells( qs_, delqs_ );

  // Loop over active (interior) cells, calculating rates and changes
  for( unsigned i=0; i<n_active_nodes_; i++ )
  {
    unsigned k = active_cell_indices_[i];
    assert( k<nnodes_ );
    eta_[k] += (b_ - delqs_[k] ) * dt_;
  }
}
```



EXAMPLE 2: OVERLAND FLOW DURING RAINSTORM

Rainfall on a flat, impermeable surface creates a shallow “mound” of water

Class definition

```
class OverlandFlowDiffWave2de : public BasicModel
{
public:
  // Constructors
  OverlandFlowDiffWave2de();

  // Overridden BMI methods
  void initialize( std::string config_file );
  void update();
  void update_until( double t );
  void finalize();
  double get_current_time() { return current_time_; }

  // Other methods
  void WriteOutputChildFormat( double time );

protected:
  vector<double> eta_;   // height of land surface, m
  vector<double> wsurf_; // water surface height, m
  vector<double> depth_; // water depth in cell, m
  vector<double> depth_at_face_;  // water depth at cell face (edge), m
  vector<double> delq_; // flow divergence, m/s
  vector<double> Sw_;     // water-surface gradient, -
  vector<double> v_;    // velocity, m/s
  vector<double> q_;    // discharge per unit width, m2/s
  vector<int> cell_ids_; // cell ID numbers
  vector<int> active_cell_indices_;  // indices of active (interior) cells
  double n_;      // roughness coefficient (Manning's n)
  double g_;       // gravitational acceleration, m/s2
  double runoff_;        // effective precipitation (runoff), m/s
  double dt_;      // time-step size, s
  double current_dt_;  // current time-step size, s
  double current_time_;  // current time in simulation, s
  ofstream depth_output_file_;     // output file (TODO: in future, use a
      generic output class?)
  ofstream water_surface_output_file_;
  ofstream velocity_output_file_;
  ofstream discharge_output_file_;

};
```

Update function

```
void OverlandFlowDiffWave2de::
update()
{
  // Calculate gradients along all edges
  mesh_->CalculateGradientsAlongEdges( wsurf_, Sw_ );

  // Calculate upstream depth values for edges
  mesh_->AssignUpstreamCellValuesToEdges( depth_, wsurf_, depth_at_face_ );

  // Calculate velocity and discharge along edges
  for( unsigned j=0; j<nedges_; j++ )
  {
    // Calculate velocity magnitude using the Manning equation
    v_[j] = (1.0/n_)*pow( depth_at_face_[j], 0.6667 )*sqrt( fabs( Sw_[j] ) );

    // Set the sign of the velocity according to the sign of the
    // water surface (downhill) slope
    v_[j] = copysign( v_[j], -Sw_[j] );

    // Calculate discharge = velocity x depth
    q_[j] = v_[j]*depth_at_face_[j];
  }

  // Calculate flux divergence
  mesh_->CalculateFluxDivergenceAtCells( q_, delq_ );

  // Loop over active (interior) cells, calculating rates and changes
  for( unsigned i=0; i<n_active_nodes_; i++ )
  {
    unsigned k = active_cell_indices_[i];
    depth_[k] += (runoff_ - delq_[k] ) * dt_;
    wsurf_[k] = eta_[k] + depth_[k];
  }
}
```