Authors: Antonia Miruna Oprescu, Wenwan Yang

In this short tutorial, you will learn how to add floating objects of different shapes to your shallow water system. In addition, you will be able to modify the shape of the ground over which the water flows while changing the water flux accordingly. Although we provide sample code for this extension, you might need to adapt the code to your own Mesh class. Here is how:

1. Sample code

You can retrieve our sample code by cloning our repository. Use the following command (1 line):

```
$ git clone
git@code.seas.harvard.edu:aoprescu_wenwanyang-cs207/aoprescu_wenwanyang-cs207.git
```

Contact us at `aoprescu@college.harvard.edu` if this doesn't work. If all went well, run the file `floating_objects.cpp`, together with your favorite `.nodes` and `.tris` files.

2. Floating objects

In this section, you will be able to add and animate multiple floating objects of different shapes.

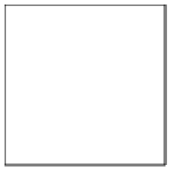
2.1. Construct objects

A floating object is defined by the force it exerts on the water (e.g.: the weight in Newtons), the position of the center and the area. Optionally, we can include motion and display features such as velocity and color. We provide the skeleton for such an object below:

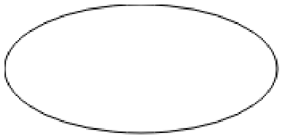
```
struct FloatingObject {
  // Force in the z-direction
  double F;
  // Object center
  Point c;
  // Parameters that determine the perimeter of the object
  double l;
  // Area computed using the shape parameters
  double A = l*l;
  //Optional: velocity and color
  Point v;
  // Color
  CS207::Color color;

  /** returns true if object is covering point p */
  bool is_over(const Point& p) const;
  /** changes the position of the center */
  void move(const double dt);
  /**
  * Returnsf orce under triangle: F/A if triangle is under boat,
  *                              0 otherwise
  */
  template<typename TRIANGLE>
  double operator()(TRIANGLE& t);
};
```

You can fill in the details and get a objects of different shapes. In the sample code, we provide the following object shapes:

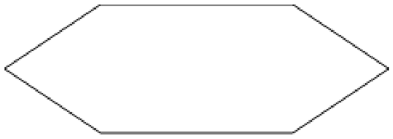
`Struct SquareObject;`



`Struct EllipticObject;`



`Struct BoatObject;`



2.2. Combine forces from multiple objects

Just like we did in previous homeworks, you can combine the forces from different sources so you can add multiple floating objects to your simulation. Here is a snippet of the sample code:

```
// Combine two templated functors
template <typename F1, typename F2>
struct Combine2Forces{
  F1 f1_;
  F2 f2_;
  Combine2Forces(F1 f1, F2 f2): f1_(f1), f2_(f2){}
  template <typename TRIANGLE>
  double operator()(TRIANGLE& t){
    return f1_(t)+f2_(t);
}
};
// Create the combined force of two
template <typename F1, typename F2>
Combine2Forces<F1, F2> make_combined_force(F1 f1, F2 f2){
  return Combine2Forces<F1, F2>(f1, f2);
}
```

where the ”forces” are actually the objects defined above. The function `make_combined_force` will add two floating objects to the simulation.

2.3. Add colors

You can get more fancy by adding different colors to the objects in the simulation. Our code can change the colors of one or more objects in the simulation by creating ”combined” colors:

```
template <typename B1, typename B2>
struct Color2Objects {
 B1 b1_;
 B2 b2_;
 Color2Objects(B1 b1, B2 b2) : b1_(b1), b2_(b2){}
template <typename NODE>
  CS207::Color operator()(const NODE& n) const {
    if(b1_.is_over(n.position()))
      return b1_.color;
    if(b2_.is_over(n.position()))
      return b2_.color;
    return CS207::Color(0.3,0.7,1.0);
     }
};

template<typename B1, typename B2>
Color2Objects<B1, B2> object_colors(B1 b1, B2 b2){
  return Color2Objects<B1, B2>(b1, b2);
}
```

This piece of code will change colors for two objects in the simulation.

3. Sources

Adding a source means changing the shape of the floor. The default shape is a plane. However, you might want for example to add a hole in the floor or an underwater mountain. Here is the skeleton of a source:

```
struct Source
{
  // returns the partial of b in respect to x
  double bx(Point p);
  // returns the partial of b in respect to y
  double by(Point p);
  // returns added flux due to source
  QVar operator()(Point p){
    return -grav * QVar(0, bx(p), by(p));
  }
};
```

We provided a structure `Pit` in the sample code that simulates a small hole in the ground with $h = -a(x^2 + y^2)$, where we set a to -0.175.

4. Putting everything together

4.1. Changes to `flux` **and** `hyperbolic\_step` functors

The flux and the `hyperbolic\_step` functions should be modified to take as arguments the objects, combined forces and colors.

Changes to `EdgeFluxCalculator`:

```
struct EdgeFluxCalculator {
  QVar operator()(double nx, double ny, double dt,
                  const QVar& qk, const QVar& qm, double pk = 0, double pm = 0) {
```

```cpp
    //...
    // Helper values
    double scale = 0.5 * n_length;
    double gh2 = 0.5 * grav * (qm.h*qm.h + qk.h*qk.h) + (pk*qm.h/ro + pm*qk.h/ro);
    //...
  }
};
```

Note that the flux functor takes in two new parameters: `pk` and `pm` which represent the prsessures exerted by the object on triangle k and triangle m, respectively. The only other change is in the canculation of `gh2`, where we added the average of the object forces on the two sides of the edge.

Changes to `hyperbolic_step`:

```cpp
1  template <typename MESH, typename FLUX, typename FORCE, typename SOURCE>
2  double hyperbolic_step(MESH& m, FLUX& f, FORCE force,
3    SOURCE& source, double t, double dt) {
4  //...
5  // Calcuate flux using adjacent triangle's Q
6      sum += f(normalvec.x, normalvec.y, dt, tri.value().q_bar, t.value().q_bar,
7        force(tri), force(t));
8  //...
9      (tri).value().q_bar =(tri).value().q_bar - (sum * dt/(tri).area()) +
10                    source(p) * (tri).value().q_bar.h * dt;
11 }
```

Note the two modifications: the flux functor takes in the force (i.e. the object structures) and applies it to both triangles (line 5), and the total flux for the triangle is added the correction due to the shape of the floor (line 9).

4.2. Add motion!

The final step requires you to built these objects in `main` and add them to the time stepping routine:

```cpp
// Object and source definition.
// Don't forget to add parameters!
SquareObject object1;
EllipticalObject object2;
BoatObject object3;
Pit source;
// time stepping
for (double t = t_start; t < t_end; t += dt) {
  object1.move(dt);
  object2.move(dt);
  object3.move(dt);

  hyperbolic_step(mesh, f,
    make_combined_force(object1,object2,object3),source, t, dt);
  post_process(mesh);

  viewer.add_nodes(mesh.node_begin(), mesh.node_end(),
    object_colors(object1, object2, object3), NodePosition(), node_map);
  }
```

The results below were obtained with `tub3.nodes`, `tub3.tris`, objects of area $A \sim 0.04$ (side $l \sim 0.2$) and weights $\sim 30\ N$.
The results are quite pretty:

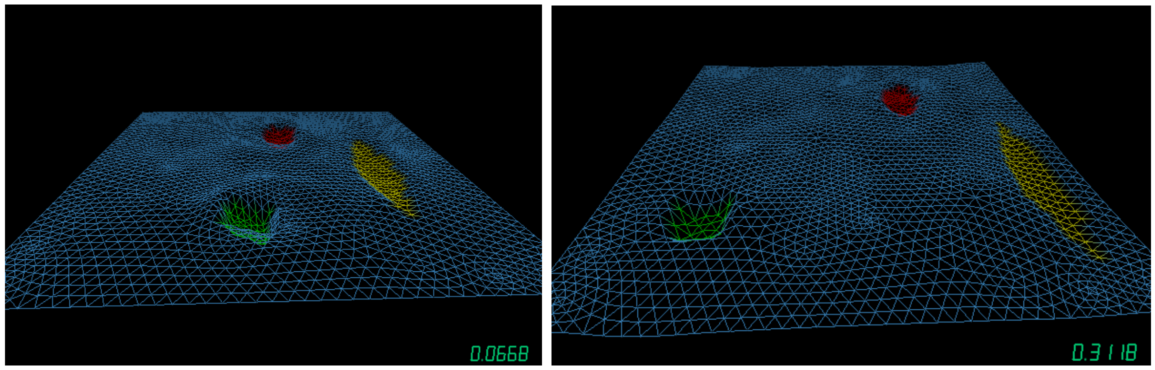


Figure 1: Three floating objects with different shapes. No source.

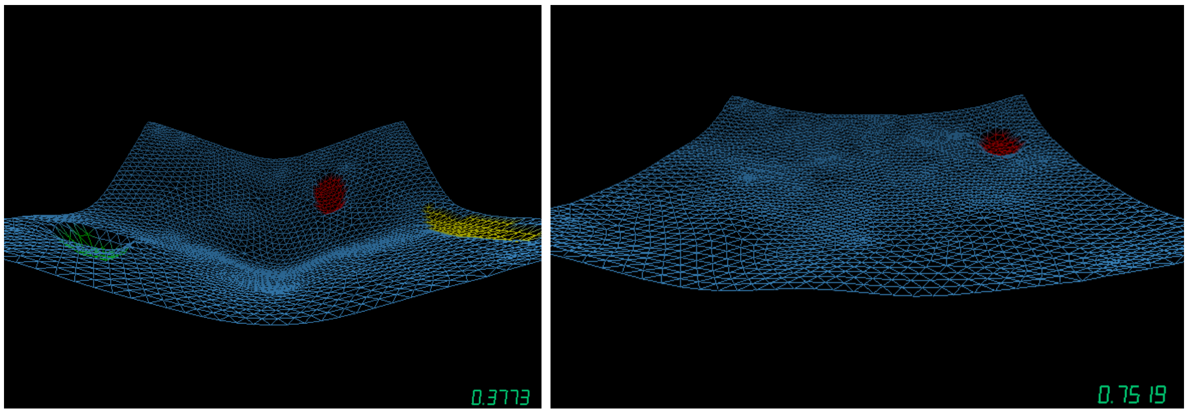


Figure 2: Hole in the ground. The water level oscillates until it reaches equilibrium.

We hope this helps!