Computer Graphics Laboratory ETH Zurich

Exercise 3: Fluid solvers

(Assigned October 23; Due November 6)

General Specifications:

The framework provided is an Eulerian 2D fluid solver, as presented in the lecture. The functions for calculating the pressure step are missing, and are to be implemented in this exercise. By the end of problem 3, the fluid solver should be functional. Please note: This is probably not the best way to code a fluid solver ☺

The program will simulate a simple smoke plume rising. You can switch the display mode by press the D key, pause by pressing Space, and use the mouse to "paint" forces into the simulation. You can change the simulation parameters in the file ″fluid2d.h″. If you want to use the png-output option on Unix-based systems, you need to install the png/pbm convert package. Image files will be created in ″./output″.

We provide a Microsoft Visual C++ project that initializes simulation settings, performs the simulation loop, and uses OpenGL/GLUT for visualization and user interaction. Makefiles for Linux and OS X are provided.

We will test the exercise by compiling and running your version of Exercise.cpp, therefore do not include external libraries, or the code will not compile in our platform.

Note: You do not need to implement the boundary conditions, the framework takes care of that.

Use central differences to derivate where needed. Consider the total grid length to be 1.0 in both x and y-direction, and calculate dx,dy appropriately. Please note that the velocities are stored on a MAC grid!!! You can assume $\rho = 1.0$.

Problem 1: Solve Poisson equation

You need to implement a simple Poisson solver for this exercise. Use the Gauss-Seidel update scheme presented in the lecture, and loop _iterations times over the grid to update all pressure values. Store the result in field. You may ignore the _accuracy parameter, or use it to optimize the process by breaking once wanted accuracy is achieved (the average error of the left hand side from the right hand side is low enough).

```
void ExSolvePoisson(int xRes, int yRes, int _iterations,
                         double _accuracy, double* field, double* b);
```

Problem 2: Correct the velocities

Implement the velocity update step (see lecture, slide 62).

```
void ExCorrectVelocities (int _xRes, int _yRes, double _dt,
                          const double* _pressure, double* _xVelocity,
                          double* _yVelocity);
```

Problem 3: Perform semi-Lagrangian advection

After having calculated the new velocities, perform semi-Lagrangian advection to all values. This means that both densities and velocities should be advected. Use bilinear interpolation to get values which are between four known points. Remember, this is a staggered grid, so pay attention to where the values are known!
In order to reduce memory allocations, you may use the pre-allocated buffers `densityTemp, xVelocityTemp, yVelocityTemp –` first compute the advected values for each cell, and store them in the temporary buffers. Only when this process is finished, copy the values back into the original buffers.

```
void ExAdvectWithSemiLagrange(int xRes, int yRes, double dt,
                              double* xVelocity, double* yVelociy,
                              double* density, double* densityTemp,
                              double* xVelocityTemp,
                              double* yVelocityTemp)
```

How to submit the exercise?

Send your version of the file Exercise.cpp by email to Amit Bermano (amit.bermano@disneyresearch.com) by November 6.