```
#ifdef __APPLE__
    #include <GLUT/glut.h>
#else
    #include <GL/glut.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


#define LATTICE_WIDTH   1000
#define LATTICE_HEIGHT  500

#define WINDOW_WIDTH    LATTICE_WIDTH
#define WINDOW_HEIGHT   LATTICE_HEIGHT

#define X_CUTOFF        5
#define FPS             4

#define VECTOR_SPACING  6

#define POINT_SIZE      6
#define SPACING         1

#define Q               9
#define DIM             2
#define BLOCK_DIM     8
#define LATTICE_SIZE LATTICE_WIDTH * LATTICE_HEIGHT * Q
#define LATTICE_LENGTH LATTICE_WIDTH * LATTICE_HEIGHT

#define OMEGA   1.5

int step = 0;
int fps = 0;

dim3 block(BLOCK_DIM, BLOCK_DIM, 1);
dim3 grid(LATTICE_WIDTH / block.x, LATTICE_HEIGHT / block.y, 1);

float lattice[LATTICE_HEIGHT * LATTICE_WIDTH * Q];
float tmp_lattice[LATTICE_HEIGHT * LATTICE_WIDTH * Q];
int boundary[LATTICE_HEIGHT * LATTICE_WIDTH];

float *gpu_lattice;
float *gpu_tmp_lattice;
```

```
int *gpu_boundary;
__constant__ int gpu_e[Q * DIM];
__constant__ float gpu_weights[Q];
__constant__ float gpu_omega;

int e[Q * DIM] = {0,0, 1,0, 0,1, -1,0, 0,-1, 1,1, -1,1, -1,-1, 1,-1};
float weights[Q] = {4.0/9.0, 1.0/9.0 , 1.0/9.0 , 1.0/9.0 , 1.0/9.0
,
                                1.0/36.0, 1.0/36.0, 1.0/36.0, 1.0/36.0};

void init_cuda( ) {
 cudaMalloc((void **) &gpu_lattice, LATTICE_SIZE);
 cudaMalloc((void **) &gpu_tmp_lattice, LATTICE_SIZE);
 cudaMalloc((void **) &gpu_boundary, LATTICE_LENGTH);

cudaMemcpy(gpu_boundary, boundary, LATTICE_LENGTH, cudaMemcpyHostToDevice);

cudaMemcpyToSymbol(gpu_e, e, Q * DIM * sizeof(int));
 cudaMemcpyToSymbol(gpu_weights, weights, Q * sizeof(float));
}

__global__ void resolve_collisions(float * gpu_lattice, float * tmp_lattice)
{
 int x = blockIdx.x * BLOCK_DIM + threadIdx.x;
 int y = blockIdx.y  * BLOCK_DIM + threadIdx.y;
    int i,j;
    int rho;
    double u[DIM];
    double u_squared;
    double u_dot_e[Q];
    double f_eq;

    double lattice[Q];

    for(i = 0; i < Q; i++) {
 lattice[i] = gpu_lattice[y * LATTICE_WIDTH + x * Q + i];
    }

    rho = 0;
    for(i = 0; i < Q; i++) {
 rho += lattice[i];
    }

    for(i = 0; i < DIM; i++) {
 u[i] = 0;
    }
```

```
 for(j = 0; j < Q; j++) {
  for(i = 0; i < DIM; i++) {
     u[i] += lattice[j] * gpu_e[j * DIM + i];
 }
    }

    u_squared = 0;
    for(i = 0; i < DIM; i++) {
 u_squared += u[i] * u[i];
    }

    for(i = 0; i < Q; i++) {
 u_dot_e[i] = 0;
    }
    for(i = 0; i < Q; i++) {
 for(j = 0; j < DIM; j++) {
     u_dot_e[i] += u[j] * gpu_e[i * DIM + j];
 }
    }

    for(i = 0; i < Q; i++) {
 f_eq = gpu_weights[i] * (rho - 1.5 * u_squared + 3 * u_dot_e[i] +
       4.5 * u_dot_e[i] * u_dot_e[i]);
 tmp_lattice[y * LATTICE_WIDTH + x * Q + i] = (1.0 - OMEGA) * lattice[i]
+
          OMEGA * f_eq;
    }
}

__global__ void propagate(float * gpu_lattice, float * gpu_tmp_lattice,
      int * gpu_boundary) {
 int x = blockIdx.x * BLOCK_DIM + threadIdx.x;
 int y = blockIdx.y  * BLOCK_DIM + threadIdx.y;
 int i, e_x, e_y, new_x, new_y, new_i;
    for(i = 0; i < Q; i++) {
    e_x = gpu_e[i * DIM];
    e_y = gpu_e[i * DIM + 1];
/*       new_x = (x + e_x + LATTICE_WIDTH) % LATTICE_WIDTH;*/
/*       new_y = (y + e_y + LATTICE_HEIGHT) % LATTICE_HEIGHT;*/
    new_x = x + e_x;
    new_y = y + e_y;
    new_i = (i + gpu_boundary[y * LATTICE_WIDTH + x]*(2 + (1 & (i*4/Q)*3)))
% Q;
    if(new_x >= 0 && new_y >= 0 && new_x < LATTICE_WIDTH &&
    new_y < LATTICE_HEIGHT) {
    if(gpu_tmp_lattice[y * LATTICE_WIDTH + x * Q + new_i] > 1.0/3.0)
```

```
{
    gpu_lattice[new_y * LATTICE_WIDTH + new_x * Q + new_i] = 1.0/3.0;
    }
    else {
    gpu_lattice[new_y * LATTICE_WIDTH + new_x * Q +new_i] =
       gpu_tmp_lattice[y * LATTICE_WIDTH + x * Q + i];
    }
    }
    }
}

void next( ) {
// cudaMemcpy(gpu_lattice, lattice, LATTICE_SIZE, cudaMemcpyHostToDe-
vice);
// cudaMemcpy(gpu_tmp_lattice, tmp_lattice, LATTICE_SIZE, cudaMemcpyHost-
ToDevice);

resolve_collisions<<<grid, block>>>(gpu_lattice, gpu_tmp_lattice);
 propagate<<<grid, block>>>(gpu_lattice, gpu_tmp_lattice, gpu_boundary);
// cudaMemcpy(lattice, gpu_lattice, LATTICE_SIZE, cudaMemcpyDeviceToHost);
// cudaMemcpy(tmp_lattice, gpu_tmp_lattice, LATTICE_SIZE, cudaMemcpyDe-
viceToHost);


}

void display( ) {
    int x, y, i, j;
    int v_x, v_y;
    double slope[DIM];
    double intensity;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity( );
    glPointSize(POINT_SIZE);

    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POINTS);
        for(y = 0; y < LATTICE_HEIGHT; y++) {
            for(x = X_CUTOFF; x < LATTICE_WIDTH; x++) {
                if(boundary[y * LATTICE_WIDTH + x]) {
                    glVertex3f(x, y, 0);
                }
            }
        }
    glEnd( );
```

```
    for (x = X_CUTOFF; x < LATTICE_WIDTH; x += VECTOR_SPACING) {
        for (y = 0; y < LATTICE_HEIGHT; y += VECTOR_SPACING) {
            intensity = 0.0;
            for(j = 0; j < DIM; j++) {
                slope[j] = 0.0;
            }
            for(v_x = x; v_x < x + VECTOR_SPACING; v_x++) {
                for(v_y = y; v_y < y + VECTOR_SPACING; v_y++) {
                    for(i = 0; i < Q; i++) {
                        for(j = 0; j < DIM; j++) {
                            intensity += lattice[v_y * LATTICE_WIDTH
+ v_x * Q + i];
                            slope[j] += e[i * DIM + j] * lattice[v_y
* LATTICE_WIDTH + v_x * Q + i];
                        }
                    }
                }
            }

            for(j = 0; j < DIM; j++) {
                slope[j] /= 0.5 * VECTOR_SPACING;
            }
            if(slope[DIM-1] < 0) {
                glColor3f(0.25 + intensity, intensity, intensity/3);
            }
            else {
                glColor3f(intensity/3, intensity, 0.25 + intensity);
            }
            glBegin(GL_LINES);
                glVertex3f(x, y, 0);
                glVertex3f(x + slope[0], y + slope[1], 0);
            glEnd( );

            glPointSize(POINT_SIZE/3);
            glVertex3f(intensity, intensity, intensity);
            glBegin(GL_POINTS);
                glVertex3f(x,y,0);
            glEnd( );
        }
    }

    glutSwapBuffers( );
}

void idle( ) {
```

```
    int i, j, k;

    for (i = 2; i < X_CUTOFF; i++) {
        for (j = LATTICE_HEIGHT/2 - X_CUTOFF; j < LATTICE_HEIGHT/2 +
X_CUTOFF; j++) {
            lattice[j * LATTICE_WIDTH + i * Q + 1] = 0.25;
/*              lattice[j][i][5] = 0.05;*/
/*              lattice[j][i][8] = 0.05;*/

            tmp_lattice[j * LATTICE_WIDTH + i * Q + 1] = 0.25;
/*              tmp_lattice[j][i][5] = 0.05;*/
/*              tmp_lattice[j][i][8] = 0.05;*/
        }
    }

    if(step || !step) {
        next( );
        step = 0;
    }
 cudaMemcpy(lattice, gpu_lattice, LATTICE_SIZE, cudaMemcpyDeviceToHost);
 cudaMemcpy(tmp_lattice, gpu_tmp_lattice, LATTICE_SIZE, cudaMemcpyDeviceToHost);
    if(fps == FPS) {
        glutPostRedisplay();
        fps -= FPS;
    }
    fps++;
}

void init_vars( )
{
    int i, j, k, l;

    for(i = 0; i < LATTICE_WIDTH; i++) {
        for(j = 0; j < LATTICE_HEIGHT; j++) {
            boundary[i * LATTICE_WIDTH + j] = 0;
            for(k = 0; k < Q; k++) {
                lattice[j * LATTICE_WIDTH + i * Q + k] = 0.10;
                tmp_lattice[j * LATTICE_WIDTH + i * Q + k] = 0.10;
            }
        }
    }

/*    for (i = 0; i < X_CUTOFF; i++) {*/
/*        for (j = 0; j < LATTICE_HEIGHT; j++) {*/
/*            for(k = 0; k < Q; k++) {*/
/*                lattice[j][i][k] = 0.2;*/
```

```
/*                  tmp_lattice[j][i][k] = 0.2;*/
/*              }*/
/*          }*/
/*      }*/

/*      for(i = LATTICE_HEIGHT/3; i < 3*LATTICE_WIDTH/4; i++) {*/
/*          for(j = LATTICE_WIDTH/3; j < 3*LATTICE_HEIGHT/4; j++)
{*/
/*              for(k = 0; k < Q; k++) {*/
/*                  lattice[j][i][k] = 0.2;*/
/*                  tmp_lattice[j][i][k] = 0.2;*/
/*              }*/
/*              lattice[j][i][1] = 0.2;*/
/*              tmp_lattice[j][i][1] = 0.2;*/
/*          }*/
/*      }*/

/*      for(i = LATTICE_HEIGHT/6; i < LATTICE_WIDTH/3; i++) {*/
/*          for(j = LATTICE_WIDTH/6; j < LATTICE_HEIGHT/3; j++)
{*/
/*              for(k = 0; k < Q; k++) {*/
/*                  lattice[i][j][k] = 1;*/
/*              }*/
/*          }*/
/*      }*/

    int x, y;
    double d_theta = 0.01;
    double theta = d_theta;
    int radius = 12;

    for(i = 0; i < 2*M_PI/d_theta; i++, theta += d_theta) {
        for(j = 0; j < radius; j++) {
            for(k = 1; k < 1; k++) {
                x = k*LATTICE_WIDTH/6 + j * cos(theta);
                y = k*LATTICE_HEIGHT/3 + j * sin(theta);
                boundary[y * LATTICE_WIDTH + x] = 1;
                for(l = 0; l < Q; l++) {
                    lattice[y * LATTICE_WIDTH + x * Q + l] = 0;
                    tmp_lattice[y * LATTICE_WIDTH + x * Q + l] = 0;
                }
            }
        }
    }

    for(i = 0; i < 2*M_PI/d_theta; i++, theta += d_theta) {
```

```
            for(j = 0; j < radius; j++) {
                x = LATTICE_WIDTH/2 + j * cos(theta);
                y = LATTICE_HEIGHT/2 + j * sin(theta);
                boundary[y * LATTICE_WIDTH + x] = 1;
                for(k = 0; k < Q; k++) {
                    lattice[y * LATTICE_WIDTH + x * Q + k] = 0;
                    tmp_lattice[y * LATTICE_WIDTH + x * Q + k] = 0;
                }
            }
        }

/*
    for(x = 0; x < LATTICE_WIDTH; x++) {
        boundary[x] = 1;
        boundary[(LATTICE_HEIGHT-1) * LATTICE_WIDTH + x] = 1;
        for(k = 0; k < Q; k++) {
            lattice[x * Q + k] = 0;
            tmp_lattice[x * Q + k] = 0;

            lattice[(LATTICE_HEIGHT-1) * LATTICE_WIDTH + x * Q +
k] = 0;
            tmp_lattice[(LATTICE_HEIGHT-1) * LATTICE_WIDTH + x *
Q + k] = 0;
        }
    }
*/

/*    for(x = 0; x < LATTICE_HEIGHT; x++) {*/
/*        boundary[x] = 1;*/
/*        boundary[x][LATTICE_WIDTH-1] = 1;*/
/*        for(k = 0; k < Q; k++) {*/
/*            lattice[x][0][k] = 0;*/
/*            tmp_lattice[x][0][k] = 0;*/

/*            lattice[x][LATTICE_WIDTH-1][k] = 0;*/
/*            tmp_lattice[x][LATTICE_WIDTH-1][k] = 0;*/
/*        }*/
/*    }*/
}

void init_gl()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
    glOrtho(0.0, WINDOW_WIDTH, 0.0, WINDOW_HEIGHT, -20.0, 20.0);

    glMatrixMode(GL_MODELVIEW);

    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    glShadeModel(GL_SMOOTH);
    glPointSize(POINT_SIZE);
    glLineWidth(POINT_SIZE);
}

void keyboard(unsigned char key, int x, int y) {
    if(key == 'n') {
        step = 1;
    }
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("D2Q9");
    glEnable(GL_DEPTH_TEST);
    init_vars();
 init_cuda( );
    init_gl();
    glutDisplayFunc(display);
    glutKeyboardUpFunc(keyboard);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0;
}
```