第 1 部分：《格子 Boltzmann 方法的理论及应用》的有关源程序

程序 2　顶盖驱动流的格子 Boltzmann 模拟

2.1 物理模型

顶盖驱动流（lid-driven flow）是计算流体与计算传热学中的一个经典问题（参见图 1），常用作等温不可压缩算法的校核算例，同时也是一个很好的格子 Boltzmann 方法入门算例。在顶盖驱动流中，方腔的上边界以一个恒定速度水平右移，而其他三个边界则保持静止不动。其基本特征是：流动稳定后，方腔的中央有一个一级大涡出现，而在左下角和右下角会分别出现一个二级涡，当雷诺数超过一临界值后，在方腔的左上角还会出现一个涡。这些涡的中心位置是雷诺数的函数。雷诺数的定义为 $Re = LU/\nu$，其中，L 是方腔的高度，U 是顶盖的移动速度，ν 是运动黏度系数。

在本附录，我们提供一个顶盖驱动流的格子 Boltzmann 模拟程序，采用 D2Q9 模型以及标准的格子 Boltzmann 方程，边界处理采用非平衡态外推格式，程序收敛判据如下：

$$E_r = \frac{\sqrt{\sum_{i,j}\left\{\left[u_x(i,j,t+\delta_t)-u_x(i,j,t)\right]^2+\left[u_y(i,j,t+\delta_t)-u_y(i,j,t)\right]^2\right\}}}{\sqrt{\sum_{i,j}\left[u_x(i,j,t+\delta_t)^2+u_y(i,j,t+\delta_t)^2\right]}} < 10^{-6}$$

模拟中，流场初始密度 $\rho_0 = 1$，顶盖驱动速度 $U = 0.1$，计算区域的大小为 $L_x \times L_y = 256 \times 256$，也即 $L = 256$；采用湿节点式布置方式（参见《格子 Boltzmann 方法的理论及应用》8.1 节），运动黏度系数由雷诺数定义式反算得到。各参数均为格子单位。

虽然该程序是针对顶盖驱动流而编写的，读者可通过改变初始条件、边界条件等，来计算其他物理问题。程序由 C++语言编写，支持的环境为 Microsoft Visual Studio 2010。

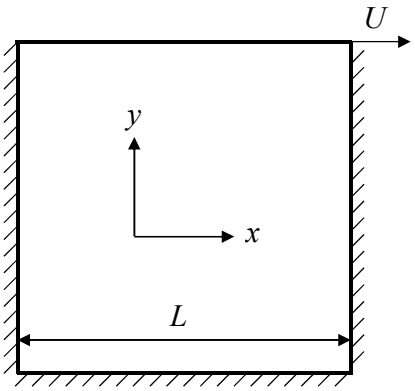


图 1 顶盖驱动流示意图

2.2 程序变量表及源程序

变量表

变量名	变量含义
Q	离散速度的总个数
Lx, Ly	x, y 方向的长度
NX+1, NY+1	x, y 方向的节点数
dx, dy	x, y 方向的网格步长
dt	时间步长
c	格子速度
n	演化次数
k	离散速度方向 α
Re	Re 数
niu	运动黏度系数 ν
tau_f	无量纲松弛时间 τ
U	顶盖速度
e[Q][2]	离散速度 $\boldsymbol{e}_\alpha$

w[Q]	权系数 ω_α
rho[NX+1][NY+1]	密度 ρ
u0[NX+1][NY+1][2]	n 时层的速度
u[NX+1][NY+1][2]	$n+1$ 时层的速度
f[NX+1][NY+1][Q]	演化前的分布函数
F[NX+1][NY+1][Q]	演化后的分布函数
rho0	流场初始密度 ρ_0
error	两个相邻时层速度的最大相对误差

源程序

```
#include"StdAfx.h"
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<iomanip>
#include<fstream>
#include<sstream>
#include<string>

using namespace std;
const int Q = 9;      //D2Q9 模型
const int NX = 256;
const int NY = 256;
const double U = 0.1;

int    e[Q][2] = {{0,0}, {1,0}, {0,1}, {-1,0}, {0,-1}, {1,1}, {-1,1}, {-1,-1}, {1,-1}};
double    w[Q] = {4.0/9, 1.0/9, 1.0/9, 1.0/9, 1.0/9, 1.0/36, 1.0/36, 1.0/36, 1.0/36};
double rho[NX+1][NY+1], u[NX+1][NY+1][2], u0[NX+1][NY+1][2], f[NX+1][NY+1][Q],
```

```
        F[NX+1][NY+1][Q];
int i, j, k, ip, jp, n;
double c, Re, dx, dy, Lx, Ly, dt, rho0, P0, tau_f, niu, error;

void init();
double feq(int k, double rho, double u[2]);
void evolution();
void output(int m);
void Error();

int main()
{
    using namespace std;
    init();
    for(n = 0;    ; n++)
    {
      evolution();
      if(n%500 == 0)
      {
        Error();
       cout<<"The "<<n<<"th computation result:"<<endl<<"The u,v of point(NX/2,NY/2) is:"
           <<setprecision(6)<<u[NX/2][NY/2][0]<<","<<u[NX/2][NY/2][1]<<endl;
       cout<<"The max relative error of uv is:"<<setiosflags(ios::scientific)<<error<<endl;
        if(n >=2000)
        {
          if(n%2000 == 0) output(n);
          if(error<1.0e-6) break;
        }
      }
    }
```

```
    return 0;
}

void init()
{
    dx = 1.0;
    dy = 1.0;
    Lx = dx*double(NY);
    Ly = dy*double(NX);
    dt = dx;
    c = dx/dt;
    rho0 = 1.0;
    Re = 1000;
    niu = U*Lx/Re;
    tau_f = 3.0*niu+0.5;
    std::cout<<"tau_f = " <<tau_f<<endl;

    for(i=0; i<=NX; i++)        //初始化
       for(j=0; j<=NY; j++)
       {
          u[i][j][0] = 0;
          u[i][j][1] = 0;
          rho[i][j] = rho0;
          u[i][NY][0] = U;
          for(k=0; k<Q; k++)
         {
             f[i][j][k] = feq(k, rho[i][j], u[i][j]);
         }
       }
}
```

```
double feq(int k,double rho,double u[2])     //计算平衡态分布函数
{
    double eu,uv,feq;
    eu = (e[k][0]*u[0]+e[k][1]*u[1]);
    uv = (u[0]*u[0]+u[1]*u[1]);
    feq = w[k]*rho*(1.0+3.0*eu+4.5*eu*eu-1.5*uv);
    return feq;
}

void evolution()
{
    for(i=1; i<NX; i++)       //演化
       for(j=1; j<NY; j++)
          for(k=0; k<Q; k++)
          {
              ip = i - e[k][0];
              jp = j - e[k][1];
              F[i][j][k] = f[ip][jp][k] + (feq(k, rho[ip][jp], u[ip][jp])-f[ip][jp][k])/tau_f;
          }

    for(i=1; i<NX; i++)     //计算宏观量
       for(j=1; j<NY; j++)
       {
          u0[i][j][0] = u[i][j][0];
          u0[i][j][1] = u[i][j][1];
          rho[i][j] = 0;
          u[i][j][0] = 0;
          u[i][j][1] = 0;
          for(k=0; k<Q; k++)
```

```
            {
                f[i][j][k] = F[i][j][k];
                rho[i][j] += f[i][j][k];
                u[i][j][0] += e[k][0]*f[i][j][k];
                u[i][j][1] += e[k][1]*f[i][j][k];
            }
            u[i][j][0] /= rho[i][j];
            u[i][j][1] /= rho[i][j];
        }

//边界处理
        for(j=1; j<NY; j++)         //左右边界
            for(k=0; k<Q; k++)
            {
                rho[NX][j] = rho[NX-1][j];
                f[NX][j][k] = feq(k, rho[NX][j], u[NX][j]) + f[NX-1][j][k] - feq(k, rho[NX-1][j], u[NX-1][j]);
                rho[0][j] = rho[1][j];
                f[0][j][k] = feq(k, rho[0][j], u[0][j]) + f[1][j][k] - feq(k, rho[1][j], u[1][j]);
            }
        for(i=0; i<=NX; i++)        //上下边界
            for(k=0; k<Q; k++)
            {
                rho[i][0] = rho[i][1];
                f[i][0][k] = feq(k, rho[i][0], u[i][0]) + f[i][1][k] - feq(k, rho[i][1], u[i][1]);

                rho[i][NY] = rho[i][NY-1];
                u[i][NY][0] = U;
                f[i][NY][k] = feq(k, rho[i][NY], u[i][NY]) + f[i][NY-1][k] - feq(k, rho[i][NY-1], u[i][NY-1]);
            }
}
```

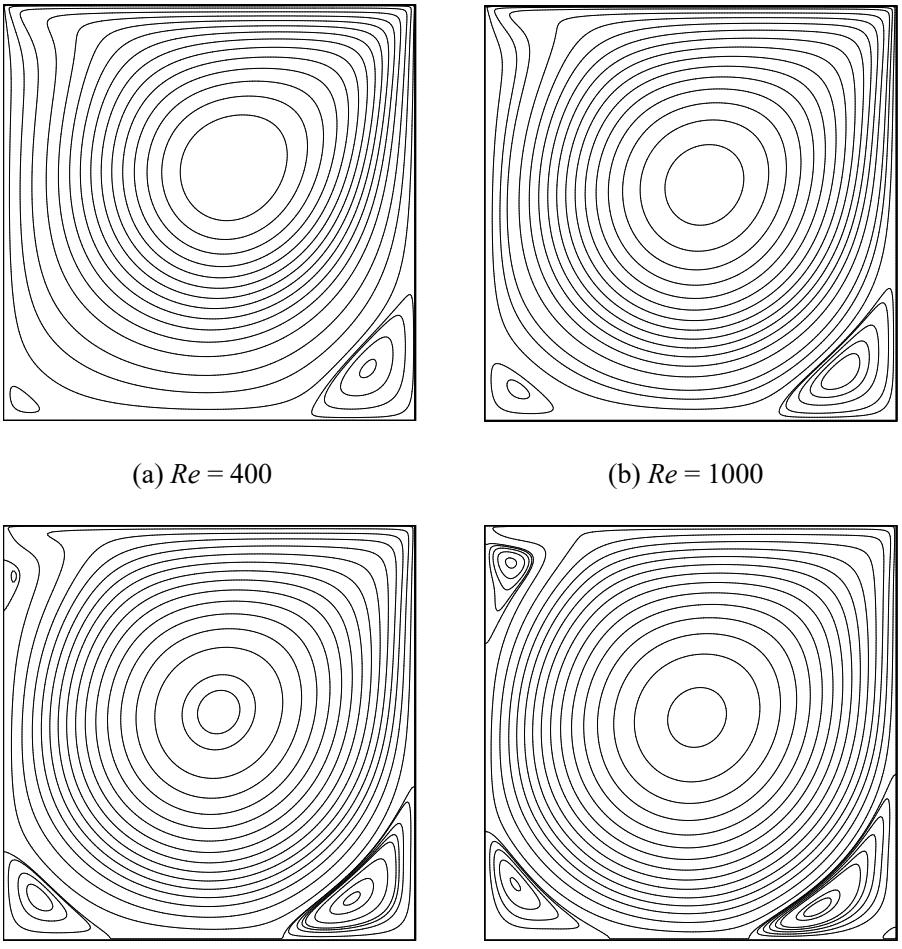
```
void output(int m)    //输出
{
    ostringstream name;
    name<<"cavity_"<<m<<".dat";
    ofstream out(name.str().c_str());
    out<<"Title =\"LBM Lid Driven Flow\"\n"<<"VARIABLES = \"X\", \"Y\", \"U\", \"V\"\n"<<"ZONE
        T=\"BOX\", I="<<NX+1<<", J="<<NY+1<<", F=POINT"<<endl;
    for(j=0; j<=NY; j++)
        for(i=0; i<=NX; i++)
        {
            out<<double(i)/Lx<<" "<<double(j)/Ly<<" "<<u[i][j][0]<<" "<<u[i][j][1]<<endl;
        }
}

void Error()
{
    double temp1, temp2;
    temp1 = 0;
    temp2 = 0;
    for(i=1; i<NX; i++)
        for(j=1; j<NY; j++)
        {
            temp1 +=
          ( (u[i][j][0]-u0[i][j][0])*(u[i][j][0]-u0[i][j][0])+(u[i][j][1]-u0[i][j][1])*(u[i][j][1]-u0[i][j][1]) );
            temp2 += (u[i][j][0]*u[i][j][0]+u[i][j][1]*u[i][j][1]);
        }
        temp1 = sqrt(temp1);
        temp2 = sqrt(temp2);
        error = temp1/(temp2+1e-30);
```

}

2.3 数值模拟结果

图 2 给出了不同雷诺数下顶盖驱动流的等流函数线，从图中可以清楚地看到雷诺数对流动模式的影响。当雷诺数较小时（ $Re \le 1000$ ），方腔中只出现三个涡：一个位于方腔中央的一级涡和一对位于左下角和右下角附近的二级涡。当 $Re = 2000$ 时，在左上角出现第三个二级涡。当雷诺数上升至 5000 时，在右下角出现了一个三级涡。从图中还可以看到，随着雷诺数的增加，一级涡的中心向方腔的中央位置移动[7]。



(a) Re = 400　　(b) Re = 1000

(c) Re = 2000　　(d) Re = 5000

图 2　顶盖驱动流的等流函数线

为了量化上述结果，我们测量了方腔中央的一级涡以及左右下角附近的两个二级涡的中心位置，结果列于表 1。表中，a、b、c 和 d 分别代表文献[8]、[9]、[10]和本程序的模拟结果。可以看到，本程序的模拟结果与其他文献的结果吻合得很好。

表 1　顶盖驱动流的涡的位置

Re		一级涡		左下涡		右下涡	
		x	*y*	*x*	*y*	*x*	*y*
400	a	0.5563	0.6000	0.0500	0.0500	0.8875	0.1188
	b	0.5547	0.6055	0.0508	0.0469	0.8906	0.1250
	c	0.5608	0.6078	0.0549	0.0510	0.8902	0.1255
	d	0.5668	0.6069	0.0471	0.0482	0.8857	0.1231
1000	a	05438	0.5625	0.0750	0.0813	0.8625	0.1063
	b	0.5313	0.5625	0.0859	0.0781	0.8594	0.1094
	c	0.5333	0.5647	0.0902	0.0784	0.8667	0.1137
	d	0.5336	0.5675	0.0820	0.0731	0.8644	0.1128
2000	a	0.5250	0.5500	0.0875	0.1063	0.8375	0.0938
	c	0.5255	0.5490	0.902	0.1059	0.8471	0.0980
	d	0.5226	0.5482	0.0863	0.1016	0.8439	0.0977
5000	a	0.5125	0.5313	0.0625	0.1563	0.8500	0.0813
	b	0.5117	0.5352	0.0703	0.1367	0.8086	0.0742
	c	0.5176	0.5373	0.0784	0.1373	0.8078	0.0745
	d	0.5158	0.5358	0.0742	0.1337	0.8055	0.0745