Palabos 中的单位
Units in Palabos

Yulan Fang

2021 年 9 月 17 日

Note. 因为 Palabos 默认下参考长度和速度为 1，所以本文中物理单位与无量纲单位数值上相同。如有错误请发邮件至 ahdhfang@hotmail.com。
Because both the reference length and time in Palabos are 1 in default, then the physical units and dimensionless units will be the same in terms of values. If there is any mistake please send me email to ahdhfang@hotmail.com.

1. 更新
Updates

第二版：1. 增加了应力的输出代码。2. 增加了英语描述。3. 修改了一些文本。
The 2nd version: 1. Add the code for outputs of the stress. 2. Add the English description. 3. Take some modifications of the contents.

2. 算例中量的转换
Conversion in Code Examples

2.1. 长度的转换
Conversion of Length

格子长度 *dx= 无量纲长度（物理长度）
Lattice length*dx=dimensionless length (physical length)

2.2. 力的转换

Conversion of force

在应用 IBM 后得到格子受力，需要定义：

After applying of the IBM the forces are obtained, it is necessary to define:

```
T forceConversion =  util::sqr(util::sqr(param.dx)) / util::sqr(param.dt);
```

转换后得到无量纲单位力。//to get the dimensionless force by conversion.

2.3. 重力的转换

Conversion of Gravity

重力作为体积力，转换为格子力可参考算例 damBreak3d.cpp 中的：

Gravity as body force, conversion can be learnt from the example case damBreak3d.cpp:

```
T gLB = 9.8 * delta_t * delta_t/delta_x;
```

2.4. 输出速度

Output Velocity

在算例的输出中，存在以下代码：

In the example cases, below codes exist:

```
vtkOut_x.writeData<3,float>(*vx, "v", param.dx / param.dt);
```

此时为 vx 乘以量纲 m，除以量纲 t，得到无量纲（或物理单位，因为默认 Palabos 的物理速度长度尺度为 1）速度。

Now the vx multiplied by the dimension m, and divided by the dimension t, to get dimensionless velocity. (Or physical velocity, because in default the referential scale is 1.)

2.5. 输出涡量

Output Vorticity

参考算例：

Example case:

```
vtkOut.writeData<3,float>(*computeVorticity(*computeVelocity(lattice)), "
    vorticity", 1./dt);
//也有
```

```
//Also
    std::unique_ptr<MultiTensorField3D<T,3> > v = computeVelocity(*lattice
    , param.outputDomain);

    vtkOut.writeData<float>(*computeNorm(*v), "velocityNorm", param.dx /
    param.dt);

    vtkOut.writeData<3,float>(*v, "velocity", param.dx / param.dt);
    std::unique_ptr<MultiTensorField3D<T,3> > vort = computeVorticity(*v);

    vtkOut.writeData<float>(*computeNorm(*vort), "vorticityNorm", 1.0 /
    param.dt);

    vtkOut.writeData<3,float>(*vort, "vorticity", 1.0 / param.dt);
```

2.6. 输出压力

Output Pressure

参考算例:

Example case:

```
vtkOut.writeData<float>( *bc.computePressure(param.boundingBox()),
                          "pressure", param.dx*param.dx/(param.dt*param.dt) );

vtkOut.writeData<float>(*boundaryCondition.computePressure(vtkDomain), "p"
    , util::sqr(dx/dt)*fluidDensity);

std::unique_ptr<MultiScalarField3D<T> > rhox = computeDensity(lattice,
    box_x);
vtkOut_x.writeData<float>(*rhox, "p", (param.dx * param.dx) / (param.dt *
    param.dt));
```

2.7. 输出应力

Output Stress

在 Palabos 中，应力输出的分量只有 6 个，因为它是对称的矩阵。

In Palabos, the stress output contains only 6 components due to the symmetricality of its matrix.

```
vtkOut.writeData<6,float>(*computeShearStress(lattice), "ShearStress", (
    util::sqr(util::sqr(dx)) / util::sqr(dt))/util::sqr(dx));
```

3. 自定义单位系统

User-defined Unit System

3.1. 粘度

Viscosity

```
//无量纲
//Dimensionless
param.nu = param.inletVelocity * param.ySide / Re;
//格子单位
//Lattice unit
T nuLB = param.nu * param.dt / (param.dx * param.dx);
param.omega = 1.0 / (DESCRIPTOR<T>::invCs2 * nuLB + 0.5);
```

4. Palabos 的 RBC 算例内的转换

Conversion in RBC case of Palabos

```
    T Cf = sp.rho_p * (sp.dx_p * sp.dx_p * sp.dx_p * sp.dx_p) / (sp.dt_p *
     sp.dt_p); // force
  T Cp = sp.rho_p * (sp.dx_p * sp.dx_p) / (sp.dt_p * sp.dt_p); // pressure
  T Ca = sp.dx_p * sp.dx_p; // area
```

5. Palabos 源码的单位

Units in source codes of Palabos

Palabos 源码包含单位的计算函数，路径位于 `src/core/units.h .cpp`。

Palabos's source codes contain the functions to calculate the units, and the path is `src/core/units.h .cpp`.

5.1. units.h

以不可压缩流体参数为例，主要代码如下：

For example, the main codes for noncompressible fluids parameter are:

```
/// Numeric parameters for isothermal, incompressible flow.
template<typename T>
class IncomprFlowParam {
public:
    /// Constructor
    /** \param latticeU_  Characteristic velocity in lattice units (
   proportional to Mach number).
     *  \param Re_ Reynolds number.
     *  \param N_  Resolution (a lattice of size 1 has N_+1 cells).
     *  \param lx_ x-length in dimensionless units (e.g. 1).
     *  \param ly_ y-length in dimensionless units (e.g. 1).
     *  \param lz_ z-length in dimensionless units (e.g. 1).
     */
    IncomprFlowParam(T physicalU_, T latticeU_, T Re_, T physicalLength_,
   plint resolution_, T lx_, T ly_, T lz_=T() )
    : physicalU(physicalU_), latticeU(latticeU_), Re(Re_), physicalLength(
   physicalLength_), resolution(resolution_), lx(lx_), ly(ly_), lz(lz_)
    { }
    //这里定义有八个参数，相比算例中定义多出了物理速度和物理长度
    //Here eight parameters are defined, compared to the example codes
   here has extra physical velocity and length definition.
    IncomprFlowParam(T latticeU_, T Re_, plint resolution_, T lx_, T ly_,
   T lz_=T() )
    : latticeU(latticeU_), Re(Re_), resolution(resolution_), lx(lx_), ly(
   ly_), lz(lz_)
    {
        physicalU      = (T)1;
        physicalLength = (T)1;
    }
    //在这里物理速度和长度被定义为1，这样的话我们可以认为物理单位与无量纲
   单位数值相同，后续只需要进行格子单位的转换。
    //Here the physical velocity and length is defined as 1, so that we
   can deem the physical units and dimensionless units are having the same
    numerical values, and subsequently only the conversion from lattice
   units is needed.
    /// velocity in lattice units (proportional to Mach number)
    //得到格子速度，latticeU即算例中定义的格子速度
    //Get the lattice velocity, latticeU is the lattice velocity that
   defined in the example codes.
```

```
  T getLatticeU() const { return latticeU; }
  /// velocity in physical units
  //得到物理速度，默认下返回的是1
  //Get the physical velocity, in default it returns 1
  T getPhysicalU() const { return physicalU; }
  /// Reynolds number
  T getRe() const      { return Re; }
  /// physical resolution
  //得到物理长度，默认为1
  //Get physical length, in default is 1.
  T getPhysicalLength() const { return physicalLength; }
  /// resolution
  //得到分辨率，相当于定义dx
  //Get the resolution, which is equivalent to define the dx
  plint getResolution() const { return resolution; }
  /// x-length in dimensionless units
  //Lx，Ly，Lz均为无量纲长度的尺寸
  //Lx，Ly，Lz are all dimensionless length
  T getLx() const      { return getPhysicalLength()*lx; }
  /// y-length in dimensionless units
  T getLy() const      { return getPhysicalLength()*ly; }
  /// z-length in dimensionless units
  T getLz() const      { return getPhysicalLength()*lz; }
  /// lattice spacing in dimensionless units
  T getDeltaX() const  { return (T)getPhysicalLength()/(T)getResolution
 (); }
  /// time step in dimensionless units
  //从物理速度中得到时间s的量纲
  //Obtain the dimension of time s from the physical velocity
  T getDeltaT() const  { return getDeltaX()*getLatticeU()/getPhysicalU()
 ; }
  /// conversion from dimensionless to lattice units for space
 coordinate
  //将无量纲长度的坐标转换为格子长度的坐标
  plint nCell(T l) const { return (int)(l/getDeltaX()+(T)0.5); }
  /// conversion from dimensionless to lattice units for time coordinate
  //将无量纲的时间坐标转换为格子单位时间坐标
  plint nStep(T t) const { return (int)(t/getDeltaT()+(T)0.5); }
  /// number of lattice cells in x-direction
  //Nx，Ny，Nz为算域的格子尺寸
```

```
    //Nx, Ny, Nz are the lattice size of the computational domain
    plint getNx(bool offLattice=false) const { return nCell(getLx())+1+(
   int)offLattice; }
    /// number of lattice cells in y-direction
    plint getNy(bool offLattice=false) const { return nCell(getLy())+1+(
   int)offLattice; }
    /// number of lattice cells in z-direction
    plint getNz(bool offLattice=false) const { return nCell(getLz())+1+(
   int)offLattice; }
    /// viscosity in lattice units
    //格子单位的粘度
    T getLatticeNu() const { return getLatticeU()*(T)getResolution()/Re; }
    /// relaxation time
    T getTau() const       { return (T)3*getLatticeNu()+(T)0.5; }
    /// relaxation frequency
    T getOmega() const     { return (T)1 / getTau(); }
private:
    T physicalU, latticeU, Re, physicalLength;
    plint resolution;
    T lx, ly, lz;
};

template<typename T>
void writeLogFile(IncomprFlowParam<T> const& parameters,
                  std::string const& title)
{
    std::string fullName = global::directories().getLogOutDir() + "plbLog.
   dat";
    plb_ofstream ofile(fullName.c_str());
    //此处定义了算例可输出的流场参数
    //Here is the definition of the outputs of fluids parameters in the
   example codes
    ofile << title << "\n\n";
    ofile << "Velocity in lattice units: u=" << parameters.getLatticeU()
   << "\n";
    ofile << "Reynolds number:           Re=" << parameters.getRe() << "\n
   ";
    ofile << "Lattice resolution:        N=" << parameters.getResolution()
    << "\n";
    ofile << "Relaxation frequency:      omega=" << parameters.getOmega()
    << "\n";
    ofile << "Extent of the system:      lx=" << parameters.getLx() << "\n";
    ofile << "Extent of the system:      ly=" << parameters.getLy() << "\n";
    ofile << "Extent of the system:      lz=" << parameters.getLz() << "\n";
    ofile << "Grid spacing deltaX:       dx=" << parameters.getDeltaX() << "\n";
    ofile << "Time step deltaT:          dt=" << parameters.getDeltaT() << "\n";
}
```

5.2. units.cpp

主要节选如下:

Main excerpts are showed below:

```
plint Units3D::lbIter(double physTime) const {
    return util::roundToInt(physTime/dt_);
}

double Units3D::physTime(plint lbIter) const {
    return lbIter*dt_;
}

plint Units3D::numCells(double physLength) const {
    return util::roundToInt(physLength/dx_);
}

double Units3D::physLength(plint numCells) const {
    return numCells*dx_;
}

double Units3D::lbVel(double physVel) const {
    return physVel * dt_/dx_;
}

double Units3D::physVel(double lbVel) const {
    return lbVel * dx_/dt_;
}

Array<double,3> Units3D::lbVel(Array<double,3> const& physVel) const {
    return physVel * dt_/dx_;
}

Array<double,3> Units3D::physVel(Array<double,3> const& lbVel) const {
    return lbVel * dx_/dt_;
}

double Units3D::lbAcceleration(double physAcceleration) const {
    return physAcceleration * dt_*dt_/dx_;
}

double Units3D::physAcceleration(double lbAcceleration) const {
    return lbAcceleration * dx_/(dt_*dt_);
}

Array<double,3> Units3D::lbAcceleration(Array<double,3> const&
    physAcceleration) const {
    return physAcceleration * dt_*dt_/dx_;
}

Array<double,3> Units3D::physAcceleration(Array<double,3> const&
    lbAcceleration) const {
    return lbAcceleration * dx_/(dt_*dt_);
}

double Units3D::lbVisc(double physVisc) const {
    return physVisc * dt_/(dx_*dx_);
}

double Units3D::physVisc(double lbVisc) const {
    return lbVisc * dx_*dx_/dt_;
}

double Units3D::tau(double physVisc, double cs2) const {
    return 0.5 + lbVisc(physVisc)/cs2;
}

```

```
double Units3D::omega(double physVisc, double cs2) const {
    return 1. / tau(physVisc, cs2);
}

double Units3D::lbPressure(double physPressure, double rho0) const {
    return physPressure / rho0 * util::sqr(dt_) / util::sqr(dx_);
}

double Units3D::physPressure(double lbPressure, double rho0) const {
    return rho0 * util::sqr(dx_)/util::sqr(dt_) * lbPressure;
}

double Units3D::lbForce(double physForce, double rho0) const {
    return physForce / rho0 * util::sqr(dt_) / util::sqr(dx_*dx_);
}

double Units3D::physForce(double lbForce, double rho0) const {
    return rho0 * util::sqr(dx_*dx_)/util::sqr(dt_) * lbForce;
}

Array<double,3> Units3D::lbForce(Array<double,3> const& physForce, double
    rho0) const {
    return Array<double,3> (
            lbForce(physForce[0], rho0),
            lbForce(physForce[1], rho0),
            lbForce(physForce[2], rho0) );
}

Array<double,3> Units3D::physForce(Array<double,3> const& lbForce, double
    rho0) const {
    return Array<double,3> (
            physForce(lbForce[0], rho0),
            physForce(lbForce[1], rho0),
            physForce(lbForce[2], rho0) );
}

double Units3D::lbTorque(double physTorque, double rho0) const {
    return physTorque / rho0 * util::sqr(dt_) / (util::sqr(dx_*dx_)*dx_);
}

```

```
double Units3D::physTorque(double lbTorque, double rho0) const {
    return rho0 * (util::sqr(dx_*dx_)*dx_)/util::sqr(dt_) * lbTorque;
}

Array<double,3> Units3D::lbTorque(Array<double,3> const& physTorque,
    double rho0) const {
    return Array<double,3> (
            lbTorque(physTorque[0], rho0),
            lbTorque(physTorque[1], rho0),
            lbTorque(physTorque[2], rho0) );
}

Array<double,3> Units3D::physTorque(Array<double,3> const& lbTorque,
    double rho0) const {
    return Array<double,3> (
            physTorque(lbTorque[0], rho0),
            physTorque(lbTorque[1], rho0),
            physTorque(lbTorque[2], rho0) );
}

Array<double,3> Units3D::lbCoord(Array<double,3> const& physCoord) const {
    return (physCoord-physDomain_.lowerLeftCorner)/dx_;
}

Array<double,3> Units3D::physCoord(Array<double,3> const& lbCoord) const {
    return physDomain_.lowerLeftCorner + lbCoord*dx_;
}
```