SiwiR 2 - Assignment 3

Dominik Bartuschat

University Erlangen-Nuremberg – System Simulation

June 16th 2015

Outline

Recapitulation of LBM

Discrete Lattice Boltzmann equation

$$f_\alpha(x + \vec{c}_\alpha \Delta t, t + \Delta t) - f_\alpha(x, t) = -\omega(f_\alpha - f_\alpha^{eq})$$

collide step

$$\tilde{f}_\alpha(x, t + \Delta t) = f_\alpha(x, t) - \omega(f_\alpha - f_\alpha^{eq})$$

stream step

$$\tilde{f}_\alpha(x + \vec{c}_\alpha \Delta t, t + \Delta t) = \tilde{f}_\alpha(x, t + \Delta t)$$

Moments of the probability functions

0. density: $\rho = \sum_\alpha f_\alpha$
1. momentum density: $\rho\vec{u} = \sum_\alpha f_\alpha \vec{c}_\alpha$

Boundary conditions

no-slip: bounce-back

$$f_{\bar{\alpha}}(x,t) = f_{\alpha}(x,t), \qquad \vec{c}_{\alpha} = -\vec{c}_{\bar{\alpha}}$$

moving no-slip: modified bounce-back

$$f_{\bar{\alpha}}(x,t) = f_{\alpha}(x,t) - 2 t_{\alpha} \rho \frac{3}{c^2} \vec{c_{\alpha}} \cdot \vec{u_w}, \qquad t_{\alpha}: \text{ weighting factor}$$

Incompressible LBM

- Standard formulation of the equilibrium distribution:

$$f_\alpha^{eq} = t_\alpha \rho \left(1 + \frac{3}{c^2} \vec{c_\alpha} \cdot \vec{u} + \frac{9}{2c^4} (\vec{c_\alpha} \cdot \vec{u})^2 - \frac{3\vec{u}^2}{2c^2} \right)$$

- Special formulation for incompressible fluids:

$$f_\alpha^{eq} = t_\alpha \left(\rho + \frac{3}{c^2} \vec{c_\alpha} \cdot \vec{u} + \frac{9}{2c^4} (\vec{c_\alpha} \cdot \vec{u})^2 - \frac{3\vec{u}^2}{2c^2} \right)$$

Incompressible LBM

For incompressible fluids it is advisable to adapt the moving wall boundary conditions:

- Compressible fluids:

$$f_{\bar{\alpha}}(x,t) = f_{\alpha}(x,t) - 2t_{\alpha}\rho\frac{3}{c^2}\vec{c_{\alpha}} \cdot \vec{u_w}$$

- Incompressible fluids ($\rho = 1$):

$$f_{\bar{\alpha}}(x,t) = f_{\alpha}(x,t) - 2t_{\alpha}\frac{3}{c^2}\vec{c_{\alpha}} \cdot \vec{u_w}$$

The VTK/Paraview Visualization

VTK File Formats

http://www.vtk.org/VTK/img/file-formats.pdf

VTK File Format

```
# vtk DataFile Version 4.0
SiwiRVisFile
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 50 50 1
ORIGIN 0 0 0
SPACING 1 1 1
POINT_DATA 2500

...
```

The VTK/Paraview Visualization

VTK File Format (cont'd)

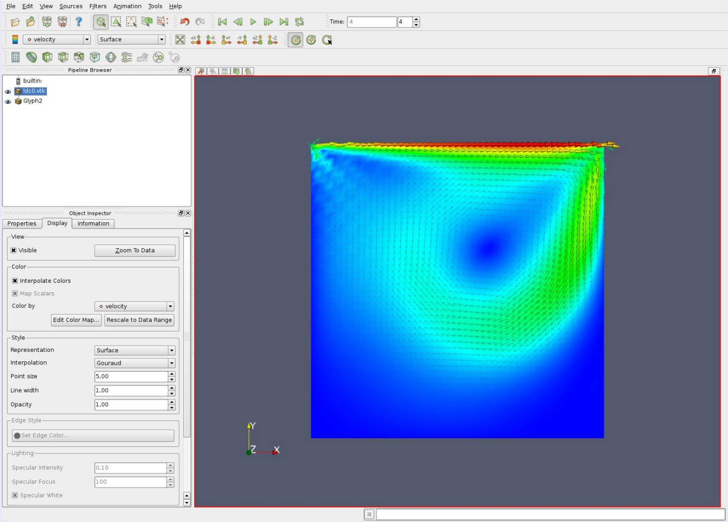
```
SCALARS flags double 1
LOOKUP_TABLE default
1
...

SCALARS density double 1
LOOKUP_TABLE default
1
...

VECTORS velocity double
-7.00552e-07 1.90304e-08 0
...
```

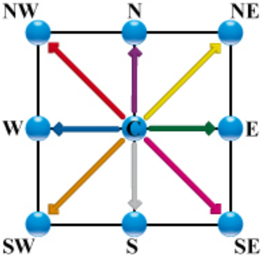
The VTK/Paraview Visualization

Paraview



Data layouts

- D2Q9-model:



- Collision-optimized data layout:

- Propagation-optimized data layout:

The Grid class

Abstraction from the actual data layout via a wrapper class

```
namespace lbm {

   // A convenient type definition
   typedef unsigned int  uint;

 ...

} // namespace lbm
```

The Grid class

```
namespace lbm {

   template< typename Type, uint Cellsize >
   class Grid
   {
    public:
      inline Grid();
      inline Grid( uint xsize, uint ysize );
      inline ~Grid();

      ...
   };

} // namespace lbm
```

The Grid class

```
public:

 ...

 inline Type& operator()( uint x, uint y, uint f );
 inline Type  operator()( uint x, uint y, uint f ) const;

 ...
```

The Grid class

```
   ...

 private:

   uint xsize_;  // Number of nodes in x-dimension
   uint ysize_;  // Number of nodes in y-dimension

   Type* data_;  // Linearized, 1-dimensional representation
                 // of the 2D data grid
};
```

The Grid class

```
// Implementation of the default constructor
template< typename Type, uint Cellsize >
Grid<Type,Cellsize>::Grid()
   : xsize_(0)
   , ysize_(0)
   , data_(0)
{}

// Implementation of the initialization constructor
template< typename Type, uint Cellsize >
Grid<Type,Cellsize>::Grid( uint xsize, uint ysize )
   : xsize_(xsize)
   , ysize_(ysize)
   , data_( new Type[Cellsize*xsize*ysize] )
{}
```

The Grid class

```
// Implementation of the function call operator
template< typename Type, uint Cellsize >
inline Type&
Grid<Type,Cellsize>::operator()( uint x, uint y, uint f )
{
   assert( x < xsize_ && y < ysize_ && f < Cellsize );
   return data_[y*xsize_*Cellsize+x*Cellsize+f];
}


// Implementation of the const function call operator

// ... Same as non-const version
```

The Grid class

```
// Partial template specialization for Cellsize = 1
template< typename Type >
class Grid<Type,1>
{
public:
   ...
   inline Type& operator()( uint x, uint y );
   inline Type  operator()( uint x, uint y ) const;
   ...
};
```

The Grid class

```
// Partial template specialization for Cellsize = 0
// No class definition => compile time error
template< typename Type >
class Grid<Type,0>;
```

The Grid class

```
// Convenient type definitions
namespace lbm {

 ...

 typedef Grid<double,9>  PDF_Field;
 typedef Grid<double,2>  V_Field;
 typedef Grid<double,1>  D_Field;
 typedef Grid<uint,1>    Flags;

 ...

} // namespace lbm
```

The Grid class

Swapping Two Grids

Due to the data dependencies in the stream step (propagation step), it is favorable to use two grids: source (`src`) and destination (`dst`). After every time step, these two grids have to be swapped.

Proper implementation of the swap functionality

- Add a `swap` **function member** to the `Grid` class
- Add an **overload** for the standard `swap` function for the `Grid` class that uses the `Grid` function member

The Grid class

The swap member function

```
namespace lbm {

   template< typename Type, uint Cellsize >
   class Grid {
    public:
      // ...
      void swap( Grid& grid ) /* throw() */ {
         std::swap( y_, grid.y_ );
         std::swap( x_, grid.x_ );
         std::swap( v_, grid.v_ );
      }
      // ...
   };

} // namespace lbm
```

The Grid class

The global swap function

```
template< typename Type, size_t N >
inline void swap( Grid<Type,N>& a,
                  Grid<Type,N>& b ) /* throw() */
{
   a.swap( b );
}
```

Use of the swap function

```
Grid<double,9> a, b;
// ... proper initialization
swap( a, b );
```

A FileReader Implementation

An Example Parameter File

```
sizex 70
sizey 80
timesteps 100
omega 1.9
vtk_file vtk/ldc.vtk
vtk_step 50
```

The Task...

... is to write a `FileReader` class that parses this parameter file, stores the parameters, and converts the parameters to the desired data type.

A FileReader Implementation

Example

```
// Parsing the parameter file
FileReader reader;
reader.readParameters( argv[1] );

// Converting the 'timesteps' parameter to a size_t value
const size_t timesteps(
   reader.getParameter<size_t>( "timesteps" )
);

// Checking the value
if( timesteps == 0 || timesteps > 1000000 ) {
   std::cerr << " Invalid 'timesteps' parameter!\n";
   return EXIT_FAILURE;
}
```

A FileReader Implementation

Task

Think about ...

- ... a suitable implementation for `FileReader`
- ... a fitting internal data structure
- ... the differences between individual data types (`int`, `unsigned int`, ...)
- ... a working implementation for a `getParameter()` function:

```
template< typename Type >
Type FileReader::getParameter(const std::string& key) c
```

Verbose Mode

Add a verbose mode to your programm such that ...

- ... it is possible to switch the verbose mode on and off very easily
- ... you make debugging easier for you
- ... the compiler can optimize away all outputs in case you switch the verbose mode off
- ... no preprocessor functionality is used

Parameterization for LBM

Normalization of basic physical values ($._p$) to the lattice parameters

$$\begin{aligned}
\Delta t &= \frac{\Delta t_p}{\Delta t_p} = 1 \\
\Delta x &= \frac{\Delta x_p}{\Delta x_p} = 1 \\
\rho &= \frac{\rho_p}{\rho_p} = 1
\end{aligned}$$

Normalization of velocity and kinematic viscosity

$u \left[\frac{m}{s}\right]$:

$$u = \frac{\Delta t_p}{\Delta x_p} \, u_p$$

$\nu \left[\frac{m^2}{s}\right]$:

$$\nu = \frac{\Delta t_p}{{\Delta x_p}^2} \, \nu_p$$

Physical parameters from lattice parameters by multiplication with inverse factors

Debugging LBM

Debugging of the lid-driven cavity

- The total mass in the system should not change, i.e. the total sum of all distribution functions should not change
- During collision, the macroscopic density and velocity of a node should not change
- The macroscopic density of a node should be close to 1 (i.e. in the range [0.9..1.1])
- The absolute value of the norm of the macrosopic velocity should not be larger than 0.1 (i.e. in the range [0..0.1]
- The individual particle distribution functions should be in the range [0..0.5]

Debugging LBM

Debugging of the lid-driven cavity

- Start with the most simple test case possible: a single lattice node surrounded by no-slip boundary nodes
- Perform obstacle / leak check: Set f_i on nodes that are never accessed to 999
- Explicitly set the macroscopic velocity via equilibrium distribution functions
  - First test case: $v = \binom{0}{0} \Rightarrow$ No changes may occur
  - Second test case: $v = \binom{0.1}{0} \Rightarrow$ Relaxation towards $\binom{0}{0}$

Summary

- implement the Lattice Boltzmann method
- use a suitable data structure
- implement a parameter reader
- visualize with paraview
- debug using a strategy