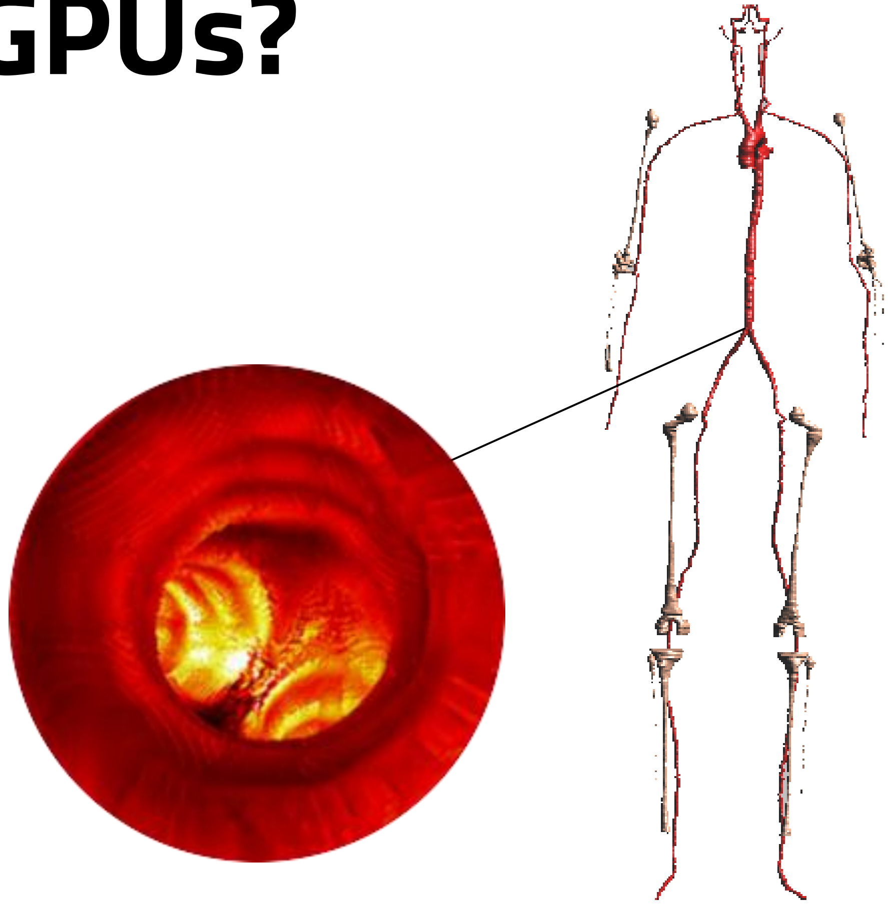
GPU-Accelerated Fluid Flow with Compute Shaders

Maciej Matyka

Institute For Theoretical Physics

Why fluids on GPUs?

- Games (**real-time**)
- Art
- Medicine



Matyka et. al., Computers & Fluids (2013)

Christ et. al., The Virtual Family. Development of anatomical CAD models of two adults and two children for dosimetric simulations, (reference)

Compute Shaders

- Run at **any** OpenGL device
- No additional **drivers**
- No **interoperability**
- Easy to port (OpenCL, CUDA)

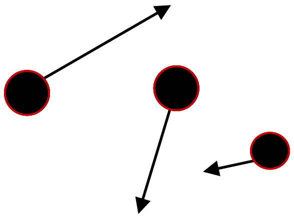
Digital fluid dynamics

Navier-Stokes

Lattice Boltzmann Method (**LBM**)

Smoothed Particle Hydrodynamics (**SPH**)

Molecular Dynamics



Ludwig Boltzmann

$$f(\mathbf{r},\mathbf{v},t)$$

Two steps algorithm

$$\underbrace{f_i(\mathbf{x}+\mathbf{c}_i, t+1) - f_i(\mathbf{x}, t)}_{\text{Transport}} = \underbrace{-\frac{1}{\tau}\left(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)\right)}_{\text{Collision}}$$

LBM basic code

```
for(int i=0 ; i < L ; i++)
for(int j=0 ; j < L ; j++)
{
  idx =  i+j*L;
  U[idx]=V[idx]=R[idx]=0;
  for(int k=0; k<9; k++)
  {
      tmp = df[ c ][ idx ][ k ];
      R[ idx ] = R[ idx ] + tmp;
      U[ idx ] = U[ idx ] + tmp * ex[ k ];
      V[ idx ] = V[ idx ] + tmp * ey[ k ];
  }
  U[idx] = U[idx]/R[idx] + fx;
  V[idx] = V[idx]/R[idx];
 // streaming + collision code here…
}
```

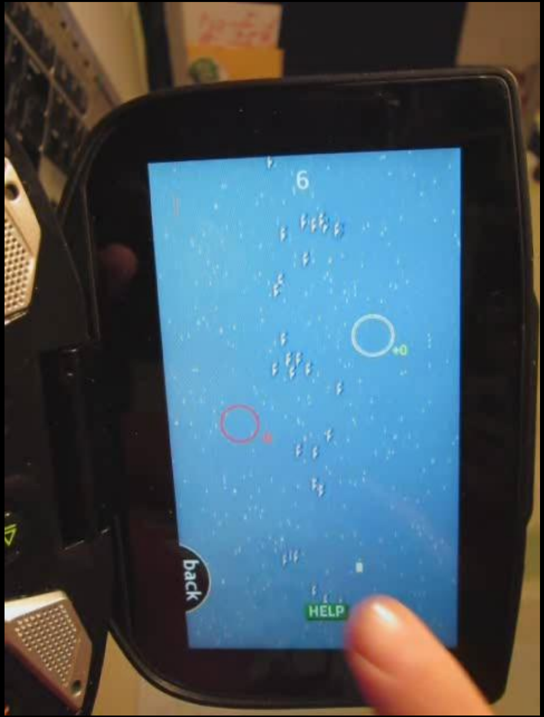
// velocity

```
 for(int k=0; k<9; k++)
{
     int ip = ( i+ex[k] + L ) % (L);
     int jp = ( j+ey[k] + L ) % (L);
     tmp = ex[k]*U[idx] + ey[k]*V[idx];
     feq = w[k] * rho * (1 – 1.5 * (U[idx]*U[idx]+V[idx]*V[idx]) + 3*tmp + 4.5 *tmp*tmp);
     if( FLAG[ip+jp*L] == 1 )
         df [1-c][idx][inv[k]] = (1-omega) * df[c][idx][k] + omega*feq
     else
         df [1-c][ip+jp*L][k] = (1-omega) * df[c][idx][k] + omega*feq;
 }
```

// transport + collision

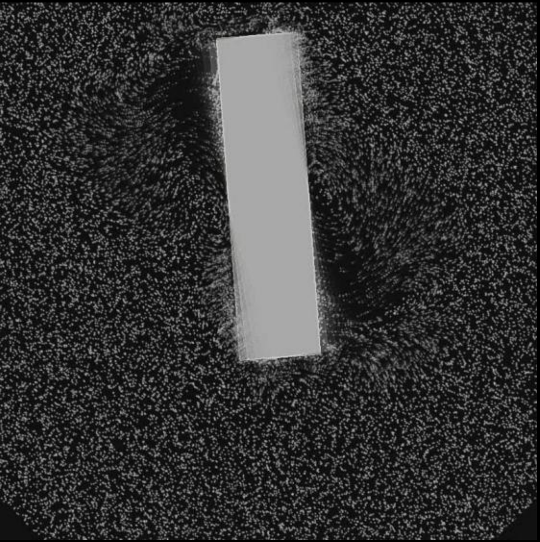
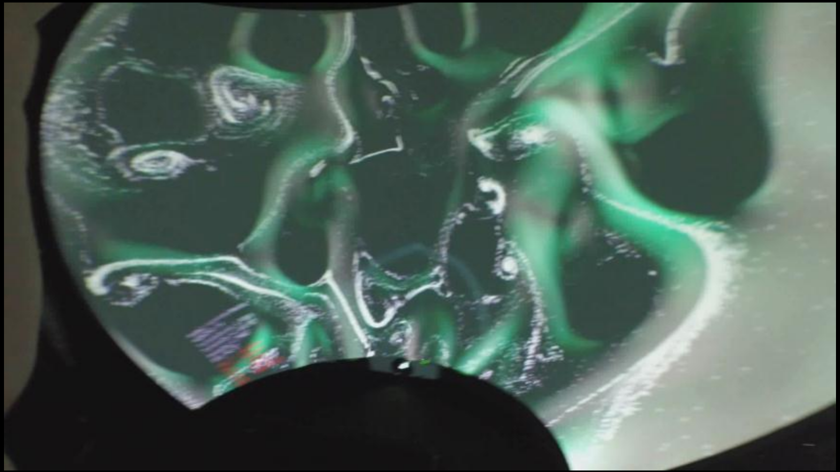
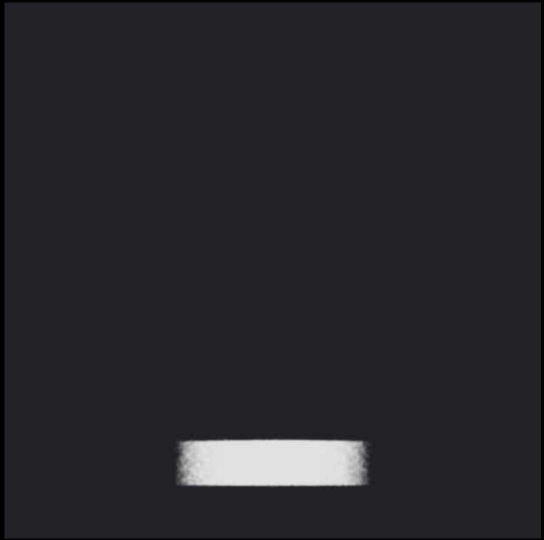
(+turbulence model by N. Delbosc, D. Chirila)

Physics Sandbox (SHIELD GIFT)

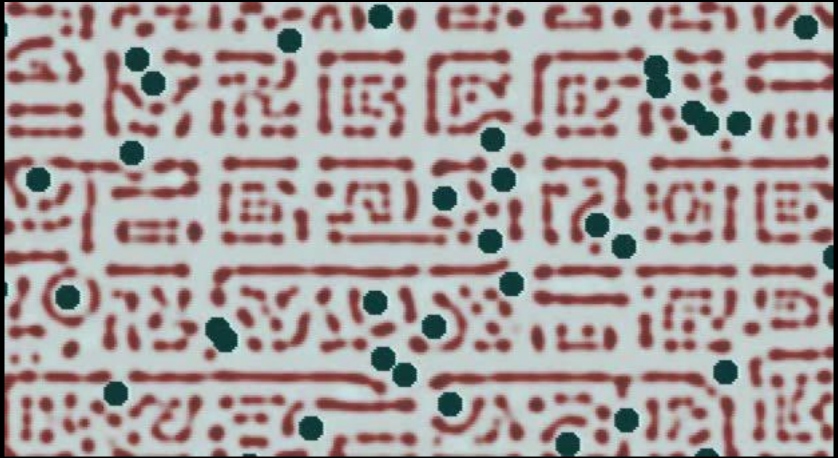


http://dreamequation.net/

ANDROID APP ON Google play

LBM




LBM on GPU

- 1 node 1 thread

- **double buffered** grid

- local (no Poisson equation)

Init shader buffer

```
glGenBuffers(...)          //  generate buffer
glBindBuffer (...)         //  bind GL_SHADER_STORAGE_BUFFER
glBufferData(...)          //   init buffer

glMapBufferRange(...)      // copy data from CPU -> GPU

glUnmapBuffer(...)         // unmap
```

Access buffer F[] from shader

```
glBindBufferBase( GL_SHADER_STORAGE_BUFFER, 2, cF_SSB );   // cpu

layout( binding=2 ) buffer dcF { int F[ ]; };              //shader
```

```
glBindBufferBase(...);                      // bind appropriate buffers

glUseProgram(...);                          // compiled shader program

glDispatchCompute( NX / 10 , NY / 10, 1);
```

Global index in LBM shader

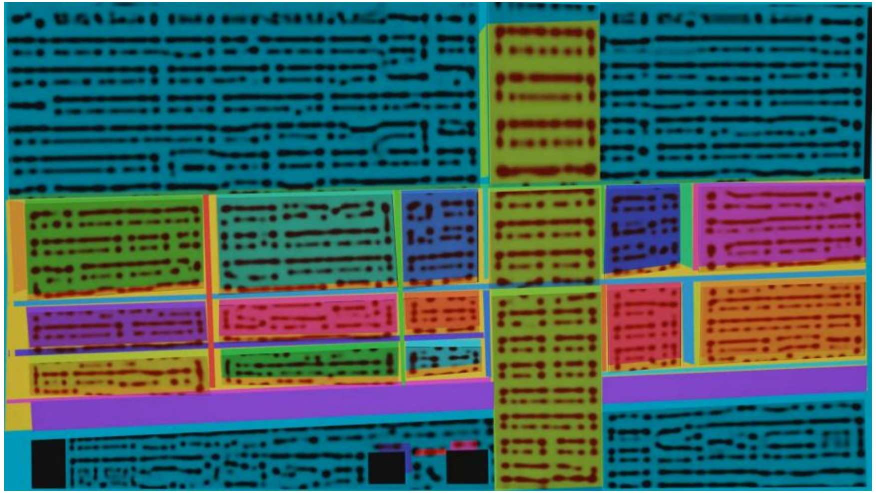
```
layout( local_size_x = 10, local_size_y = 10, local_size_z = 1 ) in;

void main()
{
        int i = int(   gl_GlobalInvocationID.x   );
        int j = int(   gl_GlobalInvocationID.y   );

        int idx = i+j*NX;    // mesh node

        ...                          // LBM step at given node
```

Video mapping installation
(collaboration: **James Hurlbut**)



Site specific **eye gaze controlled** real-time **projection mapped fluid**
The Pseudo5 **pop up art show** in North Beach, **San Francisco on May 3rd 2014**.

Starry night, Vincent Van Gogh

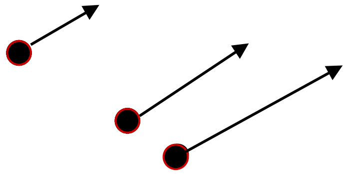
J Math Imaging Vis (2008) 30: 275–283

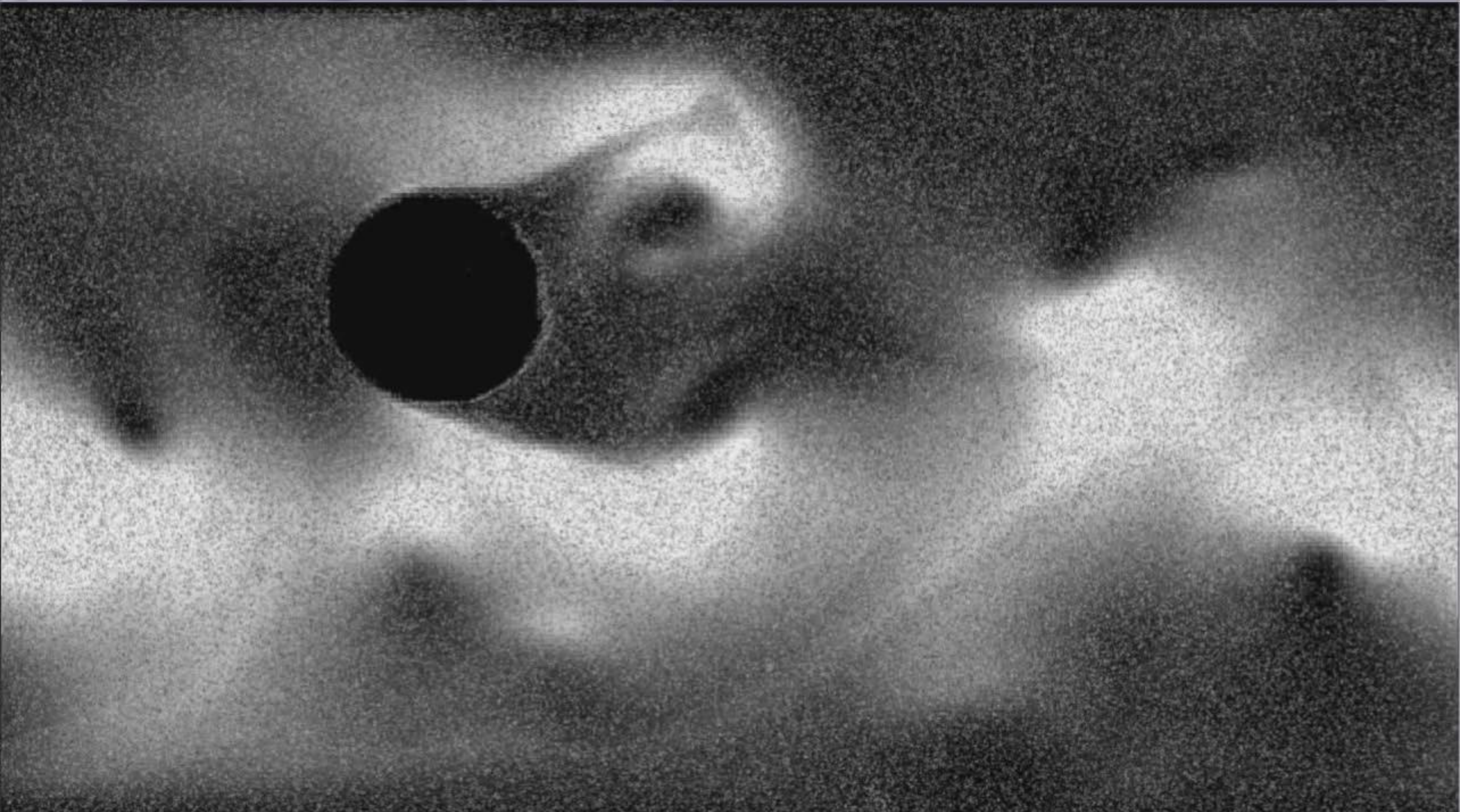
+Particles

- positions in SSB

- access to **velocity field**

- shader updates positions



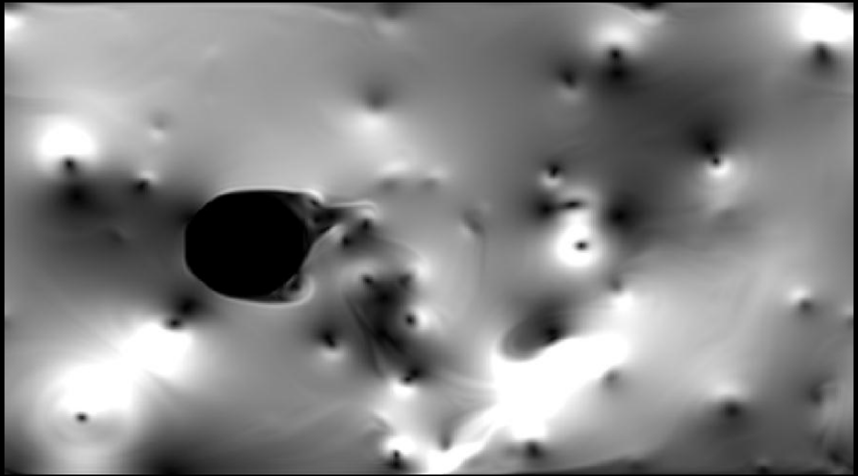


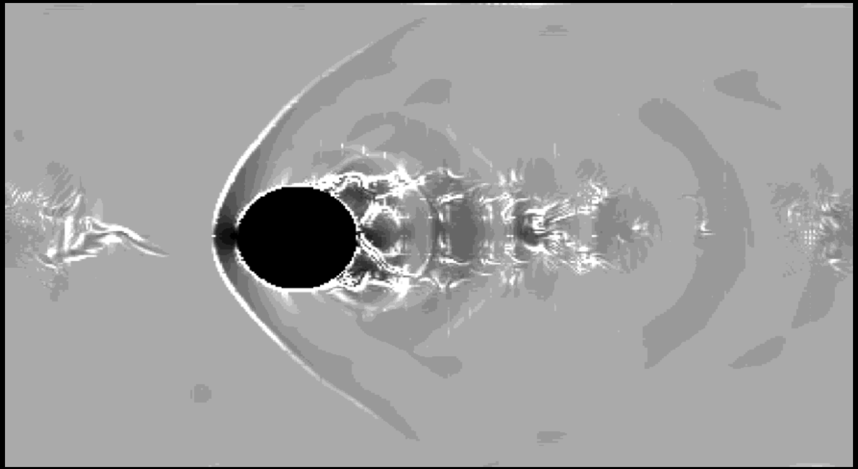
Cascaded LBM
(Collaboration: **Nicolas Delbosc**)

$$k_{17} = \lfloor \omega_{17}[(\rho/3 - 96k_9/\omega_{17} - eb - ef - nb - ne - nf - nw - sb - se - sf - sw - wb - wf - 3(nwf + nwb + nef + neb + swf + swb + sef + seb) + 2[v_x X(\omega_{17}) + v_y Y(\omega_{17}) + v_z Z(\omega_{17})] + v_x^2\{- 84k_9/\omega_{17} - b - eb - ef - f - n - ne - nw - s - se - sw - wb - wf + 2[(k_7 + k_8)/\omega_{17} - nb - neb - nef - nf - nwb - nwf - sb - seb - sef - sf - swb - swf]\} + v_y^2[- 84k_9/\omega_{17} - b - e - f - nb - ne - nf - nw - sb - se - sf - sw - w + 2(- k_7/\omega_{17} - eb - ef - neb - nef - nwb - nwf - seb - sef - swb - swf - wb - wf)] + v_z^2[- 84k_9/\omega_{17} - e - eb - ef - n - nb - nf - s - sb - sf - w - wb - wf + 2(- k_8/\omega_{17} - ne - neb - nef - nw - nwb - nwf - se - seb - sef - sw - swb - swf)] + 4[v_x v_y XY(\omega_{17}) + v_x v_z XZ(\omega_{17}) + v_y v_z YZ(\omega_{17})] + 3\rho(v_x^2 v_y^2 + v_x^2 v_z^2 + v_y^2 v_z^2))/12]\rfloor,$$

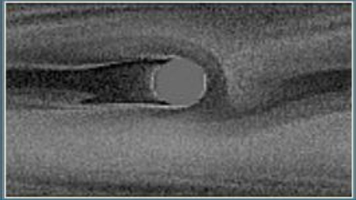
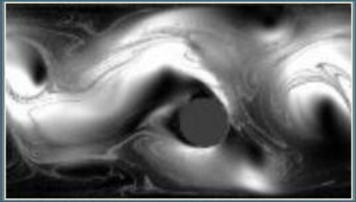
GPU's like simple algebra!

Geier M. et. al, Phys. Rev. E 73, 066705 (2006)





Codes (free): www.matyka.pl 

Example	Program	Picture	Animation	Description	Paper
Codes:	[zip]	[N/A]	[N/A]	C++ codes (OpenGL, GLUT, GLEW) for below simulations and LBM models.	[N/A]
Basic LBM	N/A		[N/A]	Compute shader GPU implementation of a standard BGK Lattice Boltzmann model. (Source code available on the website).	N/A
Smagorinsky LES LBM	[N/A]		[youtube]	Compute shader GPU implementation of a standard BGK LBM with Smagorinski LES (Lattice Boltzmann model). Source code available. Done with: Nicolas Delbosc, support Dragos Chirila.	N/A
Multiphase LBM	N/A		[youtube]	Compute shader GPU implementation of a standard BGK LBM with multiphase Shan-Chen model (Lattice Boltzmann model). Source code available.	[N/A]
Cascaded LBM	N/A		[youtube]	Compute shader GPU implementation of the Cascaded Lattice Boltzmann model. (Source code available on the website). Code mainly by Nicolas Delbosc (Cascaded LBM).	N/A

[back]

Thank you for your attention

www.matyka.pl

Thanks:

Nicolas Delbosc
James Hurlbut
Dragos Chirila
Tomasz Bednarz

Nothing below this line

----

will be shown

- Free event
- 6 talks (CFD)
- LBM, OF, GPU, Porous

LBM

on your android device (**free app**).