Folyadék szimuláció

Folyadékok

- Folyékony anyagok
- Füstszerű jelenségek
- Felhők
- Festékek

Folyadék állapota

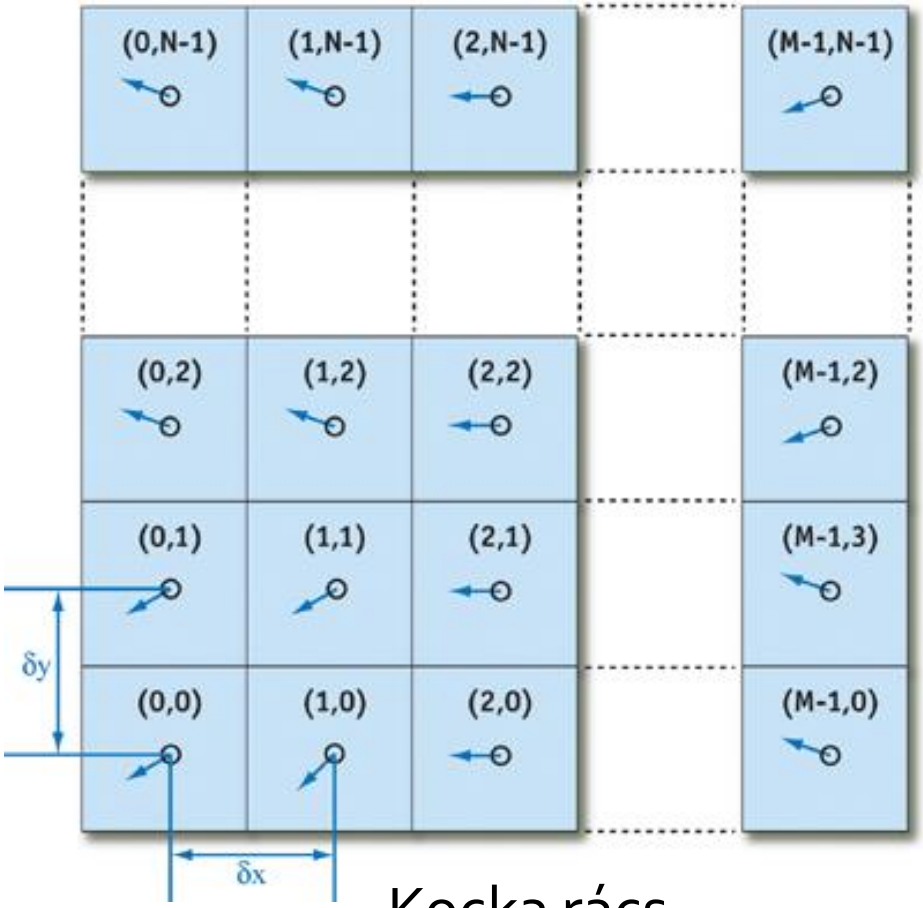
- Sebesség (vektormező)

  x = (*x*, *y*) pozíció

  u = (u,v) sebesség

  T idő

  u(**x**, *t*) = (*u*(**x**, *t*), *v*(**x**, *t*))



Kocka rács

Navier-Stokes egyenletek (1822)

- Claude Navier és George Gabriel Stokes
  - Folyékony anyagok mozgása, áramlása
- Alap feltevések
  - Az anyagban fellépő feszültség két összetevője
    - A sebesség gradiensével arányos diffúzió
    - Nyomás összetevő

Navier-Stokes egyenletek (1822)

- Számos fizikai jelenség leírására alkalmas
  - Időjárás
  - Folyadékok áramlása nem kör keresztmetszetű csatornákban
  - Repülőgépek szárnya körül fellépő áramlás
  - Szilárd testek folyékony anyagokon keresztüli mozgása (pl. a csillagok galaxisokon belül leírt mozgása)
  - Összekapcsolható a Maxwell egyenletekkel (magnetohidrodinamika)

Navier-Stokes egyenletek (1822)

- Tisztán elméleti értelemben is fontos
  - Nincs bizonyítva a három dimenziós érvényesség
  - A „létezési és simasági” probléma annyira fontos, hogy a Clay Mathematics Institute az évezred hét legfontosabb matematikai problémái között tartja számon.
    - A megoldásra egymillió dolláros díjat tűztek ki ☺

Navier-Stokes egyenletek (1822)

1.) $$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \upsilon \nabla^2 u + F$$

2.) $$\nabla \cdot u = 0$$

ρ : sűrűség

υ : viszkozitás

F : Külső erők

Összenyomhatatlan, homogén folyadékok

Az egyenlet tagjai I.

- Advekció

$$-(u \cdot \nabla)u$$

- Előre haladás, szállítás
- Bármilyen mennyiséget
- Saját vektormezőt is

Az egyenlet tagjai II.

- Nyomás

$$-\frac{1}{\rho}\nabla p$$

- Az erő nem hirtelen áramlik végig a folyadékon
- A molekulák ütköznek, nyomás keletkezik
- Gyorsulást (sebességváltozást) eredményez

Az egyenlet tagjai III.

- Diffúzió

$$\upsilon \nabla^2 u$$

- A különböző folyadékok, különbözőképpen mozognak: vannak sűrűbbek és vannak folyékonyabbak
- Viszkozitás: mennyire ellenálló a folyadék az áramlásra
- Ez az ellenállás sebesség diffúziót okoz

Az egyenlet tagjai IV.

- Külső erők

$$F$$

- Lehetnek lokálisak vagy globálisak (pl gravitáció)

Operátorok

Operátor	Definíció	Véges differencia alak
Gradiens	$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y} \right)$	$\frac{p_{i+1,j} - p_{i-1,j}}{2\delta x}, \frac{p_{i,j+1} - p_{i,j-1}}{2\delta y}$
Divergencia	$\nabla \cdot \mathrm{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$	$\frac{u_{i+1,j} - u_{i-1,j}}{2\delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\delta y}$
Laplace	$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$	$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2}$

$$\nabla^2 p = \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{(\delta x)^2}$$

Az egyenletek megoldása

- 3 egyenlet: u, v, p
- Analitikus megoldás ritkán, és csak egyszerű esetekben található
- Numerikus módszerek, inkrementális megoldás
- Ha animációt szeretnénk, az idő inkrementálás még jól is jön
- A problémát kisebb lépésekre bontjuk

(Stam, J. 1999. "Stable Fluids." In *Proceedings of SIGGRAPH 1999*)

Helmholtz-Hodge dekompozíció (projekciós lépés)

- (Bármely vektor felbontható bázisvektorok súlyozott összegére)
- Bármely vektormező felbontható vektormezők összegére :

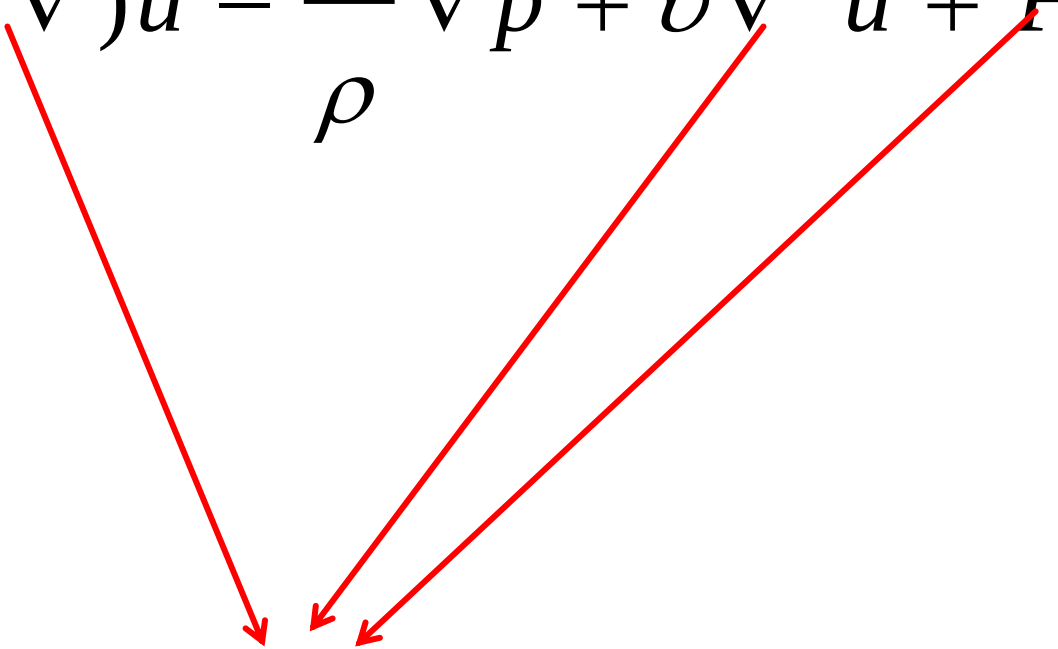
$$w = u + \nabla p,$$

$$\nabla \cdot u = 0$$

Helmholtz-Hodge dekompozíció (projekciós lépés)

1.) $$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \upsilon \nabla^2 u + F$$

2.) $$\nabla \cdot u = 0$$



$$u = w - \nabla p$$

Hogyan számítsuk ki a nyomást?

$$w = u + \nabla p \qquad / \nabla \cdot$$

$$\nabla \cdot w = \nabla \cdot (u + \nabla p)$$

$$\nabla \cdot w = \nabla \cdot u + \nabla^2 p \qquad / \nabla \cdot u = 0$$

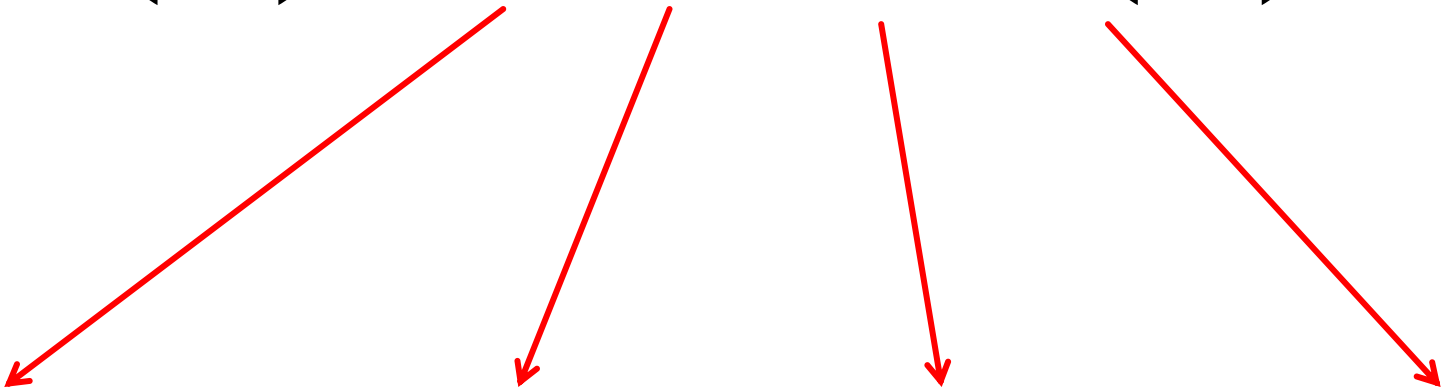
$$\nabla \cdot w = \nabla^2 p$$

Poisson egyenlet

$$(\nabla^2 x = b)$$

Mit is kell tenni egy szimulációs lépésben?

$$S(u) = P \circ F \circ D \circ A(u)$$



Projekció Külső erők Diffúzió Advekció

Advekció

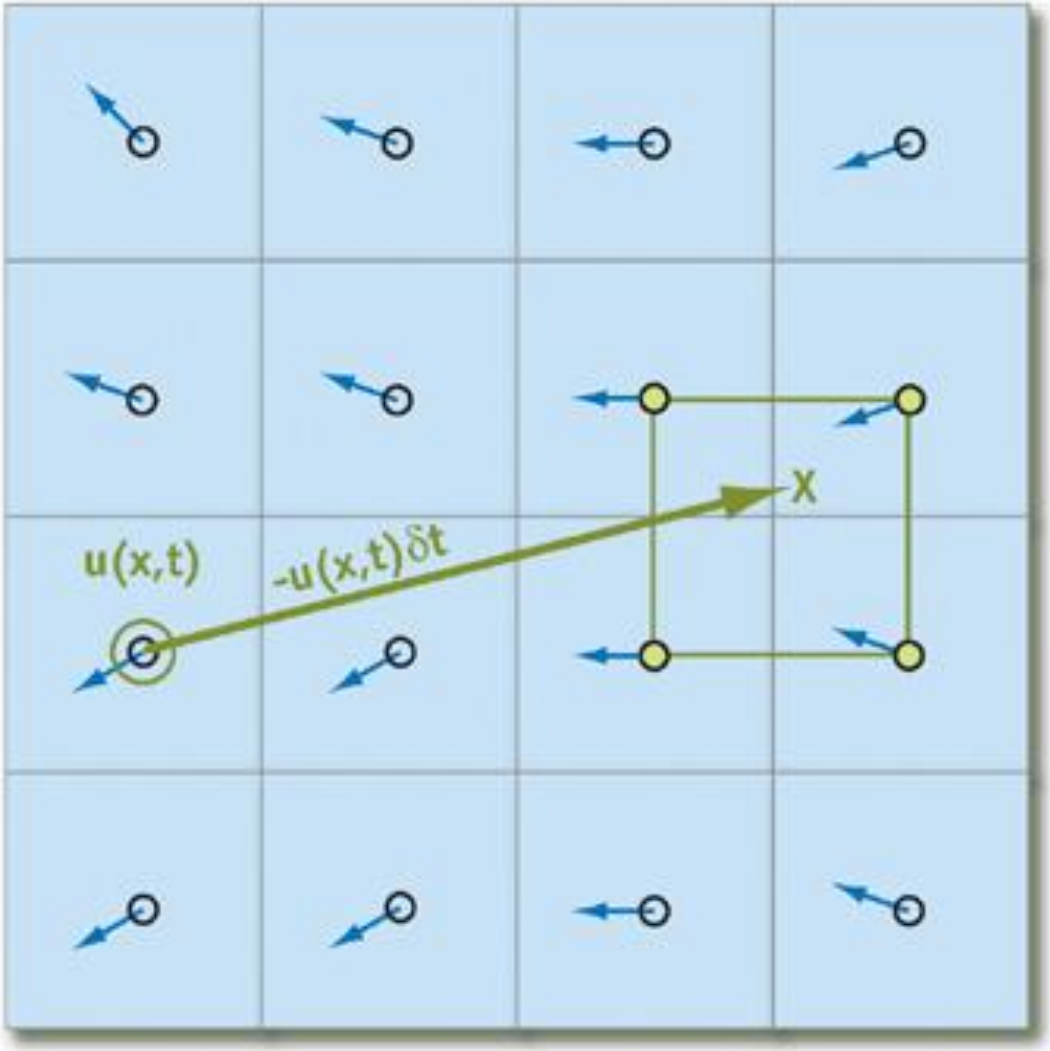
- Euler módszer, előrelépés:

$$r(t+\delta t) = r(t) + u(t)\delta t$$

- Nem stabil (és shaderből nehezen végrehajtható)
- A megoldás a visszalépés:

$$q(x, t+\delta t) = q(x - u(x,t)\delta t, t)$$

Advekció



Diffúzió

$$\frac{\partial u}{\partial t} = \upsilon \nabla^2 u$$

- Explicit megoldás:

$$u(x, t + \delta t) = u(x, t) + \upsilon \delta t \nabla^2 u(x, t)$$

- Nem stabil☹! Implicit megoldás:

$$(I - \upsilon \delta t \nabla^2) u(x, t + \delta t) = u(x, t)$$

Poisson egyenlet

$$(\nabla^2 x = b)$$

Projekció

$$u = w - \nabla p$$

$$\nabla \cdot w = \nabla^2 p$$

Poisson egyenlet

$$(\nabla^2 x = b)$$

Poisson egyenlet megoldása

- Iteratív megoldás, kiindulunk egy kezdeti állapotból és folyamatosan finomítjuk
- $Ax = b$ alakú egyenlet
- Nálunk A a Laplace operátor
- A legegyszerűbb megoldás a Jacobi iteráció

Jacobi iteráció

$$x_{i,j}^{(k+1)} = \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} + \alpha b_{i,j}}{\beta}$$

	Diffúzió	Nyomás
x	sebesség (u)	nyomás(p)
b	sebesség(u)	sebesség divergenciája $\nabla \cdot u$
α	$1/\upsilon\delta t$	-1.0
β	$1/(4+\alpha)$	0.25

Határfeltételek

- Véges tartományon számítunk, kellenek határfeltételek
- Ha az anyagot a szimulált tartományba zárjuk (falakkal vesszük körül) a sebességre és a nyomásra a feltételek:
  - Sebesség: a határokon a sebesség nulla

    (no-slip feltétel)
  - Nyomás: a határokon a nyomás változása nulla (*Neumann feltétel)*

Kiegészítés: örvénylés

- A szimuláció és a diszkretizálás numerikus hibája elmossa a mozgás bizonyos részleteit, a finom örvényeket
- Ezeket csaljuk vissza:

$$\omega = \nabla \times u \quad (curl \;\; op.)$$

$$\eta = \nabla |\omega|$$

$$\Psi = \eta / |\eta|$$

$$f_{vc} = \varepsilon (\Psi \times \omega) \delta x$$

Implementáció

- A mennyiségeket 2D tömbökben tároljuk
- Mivel a számítások során szomszédossági információk kellenek, néhány mennyiséget dupla bufferben kell tárolni (PING-PONG)
- A tömbök frissítését az OpenCL kernelek végzik
- Az egyes számítási lépésekhez külön külön kernelek szükségesek
- A megjelenítés egyszerű képernyőre rajzolás
- Kernel függvények…..(folyt)

Advekció

```
__kernel
void advection(const int gridResolution,
               __global float2* inputVelocityBuffer,
               __global float2* outputVelocityBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution – 1){

     float2 velocity = inputVelocityBuffer[id.x + id.y * gridResolution];
     float2 p = (float2)((float)id.x - dt * velocity.x, (float)id.y - dt * velocity.y);
     outputVelocityBuffer[id.x + id.y * gridResolution] = getBil(p, gridResolution,
                                                          inputVelocityBuffer);
  }
  else{ //határfeltételek
     if(id.x == 0) outputVelocityBuffer[id.x + id.y * gridResolution] =
        - inputVelocityBuffer[id.x + 1 + id.y * gridResolution];
     ...
  }
}
```

Divergencia

```
__kernel
void divergence(const int gridResolution,
                __global float2* velocityBuffer,
                __global float* divergenceBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution - 1){

     float2 vL = velocityBuffer[id.x - 1 + id.y * gridResolution];
     float2 vR = velocityBuffer[id.x + 1 + id.y * gridResolution];
     float2 vB = velocityBuffer[id.x + (id.y - 1) * gridResolution];
     float2 vT = velocityBuffer[id.x + (id.y + 1) * gridResolution];

     divergenceBuffer[id.x + id.y * gridResolution] =
               0.5f * ((vR.x - vL.x) + (vT.y - vB.y));
  }
  else{
     divergenceBuffer[id.x + id.y * gridResolution] = 0.0f;
  }
}
```

Nyomás számítása, Jacobi iteráció

```
__kernelvoid pressureJacobi(const int gridResolution,
                            __global float* inputPressureBuffer,
                            __global float* outputPressureBuffer,
                            __global float* divergenceBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution - 1){

    float alpha = -1.0f;
    float beta = 0.25f;
    float vL = inputPressureBuffer[id.x - 1 + id.y * gridResolution];
    float vR = inputPressureBuffer[id.x + 1 + id.y * gridResolution];
    float vB = inputPressureBuffer[id.x + (id.y - 1) * gridResolution];
    float vT = inputPressureBuffer[id.x + (id.y + 1) * gridResolution];
    float divergence = divergenceBuffer[id.x + id.y * gridResolution];

    outputPressureBuffer[id.x + id.y * gridResolution] =
          (vL + vR + vB + vT + alpha * divergence) * beta;
  }else{ //határfeltételek
    if(id.x == 0) outputPressureBuffer[id.x + id.y * gridResolution] =
                  inputPressureBuffer[id.x + 1 + id.y * gridResolution]; ...}
}
```

Projekció

```
__kernel
void projection(const int gridResolution,
                __global float2* inputVelocityBuffer,
                __global float* pressureBuffer,
                __global float2* outputVelocityBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution - 1){

    float pL = pressureBuffer[id.x - 1 + id.y * gridResolution];
    float pR = pressureBuffer[id.x + 1 + id.y * gridResolution];
    float pB = pressureBuffer[id.x + (id.y - 1) * gridResolution];
    float pT = pressureBuffer[id.x + (id.y + 1) * gridResolution];
    float2 velocity = inputVelocityBuffer[id.x + id.y * gridResolution];

    outputVelocityBuffer[id.x + id.y * gridResolution] =
          velocity – (float2)(pR - pL, pT - pB);
  }
  else {//határfeltételek
    if(id.x == 0) outputVelocityBuffer[id.x + id.y * gridResolution] =
         -inputVelocityBuffer[id.x + 1 + id.y * gridResolution];          ...
  }
}
```

Diffúzió

```
__kernel
void diffusion(const int gridResolution,
               __global float2* inputVelocityBuffer,
               __global float2* outputVelocityBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  float viscousity = 0.01f;
  float alpha = 1.0f / (viscousity * dt);
  float beta = 1.0f / (4.0f + alpha);
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution - 1){

    float2 vL = inputVelocityBuffer[id.x - 1 + id.y * gridResolution];
    float2 vR = inputVelocityBuffer[id.x + 1 + id.y * gridResolution];
    float2 vB = inputVelocityBuffer[id.x + (id.y - 1) * gridResolution];
    float2 vT = inputVelocityBuffer[id.x + (id.y + 1) * gridResolution];
    float2 velocity = inputVelocityBuffer[id.x + id.y * gridResolution];

    outputVelocityBuffer[id.x + id.y * gridResolution] =
             (vL + vR + vB + vT + alpha * velocity) * beta;
  } else {
    outputVelocityBuffer[id.x + id.y * gridResolution] =
          inputVelocityBuffer[id.x + id.y * gridResolution];
  }}
```

Örvénylés

```
__kernel
void vorticity(const int gridResolution,
               __global float2* velocityBuffer,
               __global float* vorticityBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution – 1){

    float2 vL = velocityBuffer[id.x - 1 + id.y * gridResolution];
    float2 vR = velocityBuffer[id.x + 1 + id.y * gridResolution];
    float2 vB = velocityBuffer[id.x + (id.y - 1) * gridResolution];
    float2 vT = velocityBuffer[id.x + (id.y + 1) * gridResolution];

    vorticityBuffer[id.x + id.y * gridResolution] =
            (vR.y - vL.y) - (vT.x - vB.x);
  }
  else{
    vorticityBuffer[id.x + id.y * gridResolution] = 0.0f;
  }
}
```

Sebesség az örvénylésből

```
__kernel
void addVorticity(const int gridResolution,
                  __global float* vorticityBuffer,
                  __global float2* velocityBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  const float scale = 0.2f;
  if(id.x > 0 && id.x < gridResolution - 1 &&
     id.y > 0 && id.y < gridResolution - 1){

    float vL = vorticityBuffer[id.x - 1 + id.y * gridResolution];
    float vR = vorticityBuffer[id.x + 1 + id.y * gridResolution];
    float vB = vorticityBuffer[id.x + (id.y - 1) * gridResolution];
    float vT = vorticityBuffer[id.x + (id.y + 1) * gridResolution];
    float4 gradV = (float4)(vR - vL, vT - vB, 0.0f, 0.0f);
    float4 z = (float4)(0.0f, 0.0f, 1.0f, 0.0f);

    if(dot(gradV, gradV)){
      float4 vorticityForce = scale * cross(gradV, z);
      velocityBuffer[id.x + id.y * gridResolution] += vorticityForce.xy * dt;
    }
  }
}
```

Külső erők

```
__kernel
void addForce(const float x,
              const float y,
              const float2 force,
               const int gridResolution,
              __global float2* velocityBuffer,
               const float4 density,
              __global float4* densityBuffer)
{
  int2 id = (int2)(get_global_id(0), get_global_id(1));
  float dx = ((float)id.x / (float)gridResolution) - x;
  float dy = ((float)id.y / (float)gridResolution) - y;

  float radius = 0.001f;

  float c = exp( - (dx * dx + dy * dy) / radius ) * dt;

  velocityBuffer[id.x + id.y * gridResolution] += c * force;
  densityBuffer[id.x + id.y * gridResolution] += c * density;
}
```

Kiegészítési lehetőségek

- Felhajtóerő és gravitáció

$$f_{buoy} = (-\kappa d + \sigma(T - T_0))\,\hat{j}$$

- Termodinamikai szimuláció (felhők)
- 3 dimenzióban
- Más rács típusok: a vektormezőkre FCC
- Tömör testekkel való interakció (voxelizálás, határfeltételek kezelése)

Együttműködés a grafikus API-val

- Célja az átjárás megteremtése
  - OpenGL és DirectX támogatás
  - Megoszthatóak
    - Általános buffer objektumok (pl. vertex buffer)
    - Textúrák
    - Render bufferek
  - A megosztandó objektumokat a grafikus API hozza létre
    - OpenCL-beli használat előtt zárolni kell
  - Az objektum használata kizárólagos!

Együttműködés a grafikus API-val

- OpenGL és OpenCL kontextus megosztás
  - GL_SHARING_EXTENSION
  - OpenGL kontextus információk

```
cl_int
clGetGLContextInfoKHR(const cl_context_properties *props,
                      cl_gl_context_info param_name,
                      size_t param_value_size,
                      void* param_value,
                      size_t* param_value_size_ret)
```

  - CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR
  - CL_DEVICES_FOR_GL_CONTEXT_KHR

Együttműködés a grafikus API-val

- OpenGL és OpenCL kontextus megosztás
  - OpenCL kontextus létrehozás

```
cl_context
clCreateContext(const cl_context_properties *props,
                                cl_uint num_devices,
                                const cl_device_id *devices,
                                void (*pfn_notify)(...),
                                void *user_data,
                                cl_int *errcode_ret)
```

    - Tulajdonságok:
      - CL_GL_CONTEXT_KHR: OpenGL kontextus
      - CL_WGL_HDC_KHR: az OpenGL kontextus HDC-je
      - CL_CONTEXT_PLATFORM: platform_id

Együttműködés a grafikus API-val

- Kontextus megosztása

```
InitGL();
cl_platform platform = createPlatform();
cl_device_id device_id = createDevice(platform, CL_DEVICE_TYPE_GPU);
cl_context sharedContext = 0;

if(CheckSharingSupport(device_id)){
  cl_context_properties props[] = {
    CL_GL_CONTEXT_KHR, (cl_context_properties)wglGetCurrentContext(),
    CL_WGL_HDC_KHR, (cl_context_properties)wglGetCurrentDC(),
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
    0
  };

  sharedContext =
            clCreateContext(props, 1, &device_id, NULL, NULL, &err);
}
```

Együttműködés a grafikus API-val

- Buffer objektumok megosztása

```
cl_mem clCreateFromGLBuffer(cl_context context,
                            cl_mem_flags flags,
                            GLuint bufobj,
                            cl_int* errcode_ret)
```

- Image objektumok megosztása

```
cl_mem clCreateFromGLTexture2D(cl_context context,
                               cl_mem_flags flags,
                               GLenum texture_target,
                               GLint miplevel,
                               GLuint texture,
                               cl_int* errcode_ret)
```

Együttműködés a grafikus API-val

- Render buffer megosztása

```
cl_mem clCreateFromGLRenderBuffer(cl_context context,
                                  cl_mem_flags flags,
                                  GLuint renderbuffer,
                                  cl_int* errcode_ret)
```

- Az OpenCL objektumok tulajdonságai
  - Létrehozáskor aktuális értékek alapján
  - Nem követik az OpenGL objektum változásait!
    - Amennyiben változik újra meg kell osztani!

Együttműködés a grafikus API-val

- Buffer objektum megosztása
  - OpenGL vertex buffer mint OpenCL memória objektum

```
GLuint vbo;

glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);

cl_mem vboCL;

vboCL = clCreateFromGLBuffer(sharedContext, CL_MEM_WRITE_ONLY,
                             vbo, NULL);
```

Együttműködés a grafikus API-val

- Objektum lefoglalása

```
cl_int clEnqueueAcquireGLObjects(cl_command_queue command,
                                 cl_uint num_objects,
                                 const cl_mem* mem_objects,
                                 ...)
```

- Objektum felszabadítása

```
cl_mem clEnqueueReleaseGLObjects(cl_command_queue command,
                                 cl_uint num_objects,
                                 const cl_mem* mem_objects,
                                 ...)
```

  - Minden használat előtt le kell foglalni
  - Használat után fel kell szabadítani

Együttműködés a grafikus API-val

- Szinkronizáció OpenGL és OpenCL között
  - Nincs explicit szinkronizáció!
    - Szüksége lenne mindkét API támogatására
  - Mindkét API oldalán a csővezeték kiürítése
    - OpenGL: glFinish()
    - OpenCL: clFinish()
  - Implementáció függően más megoldás is lehet
    - glFlush() és clEnqueueBarrier()

Együttműködés a grafikus API-val

- Buffer objektum használata
  - OpenGL vertex buffer mint OpenCL memória objektum

```
// OpenGL hívások
glFinish();

clEnqueueAcquireGLObjects(command, 1, &vboCL,
                          0, NULL, NULL);

// Kernel paraméterek beállítása és kernel végrehajtás
clFinish();

clEnqueueReleaseGLObjects(commands, 1, &vboCL,
                          0, NULL, NULL);

// OpenGL hívások
```