# ANIMATION PHYSIQUE D'UNE SURFACE D'EAU

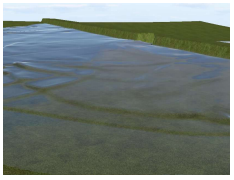
Equation de Saint-Venant ou Shallow Water Equation
TP : OpenGL

1

## Equation de Saint-Venant

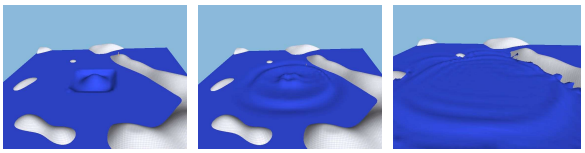
Equations de Saint-Venant ou Shallow Water equations

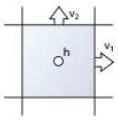
- Dérivée des eq. de Navier-Stokes
- Real Time Physics, Siggraph 2008 class.
- décrit les écoulements naturels : surface libre et eau peu profonde
- Grille 2D Approche Eulerienne (!= particule, approche Lagrangienne)



2

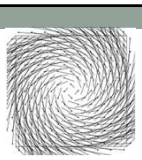
## Equation de Saint-Venant





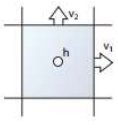

3

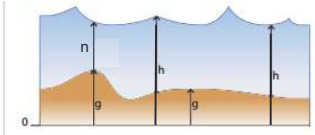
## Equation de Saint-Venant



Champ de vecteur v

- Grille 2D
- h hauteur du fluide
- g hauteur du sol
- n hauteur du fluide sur le sol : n = h−g.
- $v(v_1,v_2)$ vitesse du fluide dans le plan horizontal




ATTENTION : dans le papier original, le schéma avec h n'est pas juste

4

## Préambule

- Opérateur gradient = vecteur 2D $\vec{\nabla} = \left(\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y}\right)$

Sur une grille régulière par différence finie

| -1 | 0 | 1 |

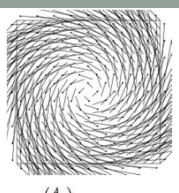
| -1 |
|---|
| 0 |
| 1 |

- Operateur Div = Divergeance d'un champ de vecteurs
  - somme les termes du gradient = scalaire

$$\nabla \cdot \boldsymbol{A} \equiv \operatorname{div} \boldsymbol{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y}$$

5

## Divergence



En dimension 3 et en coordonnées cartésiennes, la divergence d'un champ de vecteurs $\vec{A} = \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$ a pour expression[1]

$$\operatorname{div} \vec{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \frac{\partial A_z}{\partial z}$$

Formellement, l'opérateur divergence appliqué à un champ vectoriel $\vec{A}$ peut s'interpréter comme produit scalaire du vecteur nabla $\vec{\nabla}$ par le vecteur $\vec{A}$.

$$\vec{\nabla} \cdot \vec{A} = \operatorname{div} \vec{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \frac{\partial A_z}{\partial z}$$

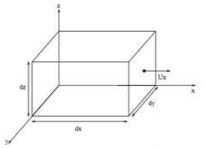
6

## Divergence

Le flux de $\vec{u}$ sortant de la face de droite dans la direction $x$ est $u_x(x+dx,y,z)dydz$.

Le flux de $\vec{u}$ entrant par la face de gauche dans la direction $x$ est $-u_x(x,y,z)dydz$.

Le bilan de flux $[u_x(x+dx,y,z)-u_x(x,y,z)]dydz = \frac{\partial u_x}{\partial x}dxdydz = \frac{\partial u_x}{\partial x}dV$



Le même raisonnement peut être fait dans la direction $y$ et dans la direction $z$. Le bilan de flux au travers des faces du volume peut donc s'écrire

$$d\phi = \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z}\right)dV = (\vec{\nabla}.\vec{u})dV.$$

La divergence d'un champ vectoriel $\vec{u}$ est un scalaire défini par :

$$div(\vec{u}) = \vec{\nabla}.\vec{u} = \frac{\partial ux}{\partial x} + \frac{\partial uy}{\partial y} + \frac{\partial uz}{\partial z}.$$

7

## Particules : approche Lagrangienne

(rappel) TP particules

- Mise à jour vitesse $v(t+dt) = v(t) + \frac{F}{m}.dt$
- Mise à jour position $p(t+dt) = p(t) + v(t).dt$

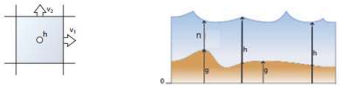
8

## Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)
- Approche Eulerienne

$$\frac{\partial \eta}{\partial t} + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1) \text{ avec } \nabla\cdot\boldsymbol{A} \equiv \mathrm{div}\boldsymbol{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y}$$
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla\mathbf{v})\mathbf{v} = a_n\nabla h\,, \quad (2)$$

- $a_n$ : accélération verticale du fluide (gravité)=9.81
- (1) s'occupe de la variation de quantité d'eau
- (2) s'occupe de la variation de la vitesse



9

## Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)
- Approche Eulerienne

$$\frac{\partial \eta}{\partial t} + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1) \text{ avec } \nabla\cdot\boldsymbol{A} \equiv \mathrm{div}\boldsymbol{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y}$$
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla\mathbf{v})\mathbf{v} = a_n\nabla h\,, \quad (2)$$

- $a_n$ : accélération verticale du fluide (gravité)=9.81
- (1) s'occupe de la variation de quantité d'eau
- (2) s'occupe de la variation de la vitesse
- Partie gauche : calculer par advection

10

## Saint-Venant : intégration

- Shallow-water-step(h,v,g)

(1) n = Advect(n,v) // partie gauche de eq (1)

(2) v1 = Advect(v1,v) // partie gauche de eq (2)
(3) v2 = Advect(v2,v) // partie gauche de eq (2)

(4) n' = Update-height(n,v)
(5) h = n'+g

(6) Update-velocities(h,v1,v2)
(7) n=n'

11

## Advection

Wikipedia :

L'**advection** est le transport d'une quantité (scalaire ou vectorielle) d'un élément donné (tel que la chaleur, l'énergie interne, un élément chimique, des charges électriques) par le mouvement (et donc la vitesse) du milieu environnant.

Donc l'advection correspond au transport d'une quantité (scalaire ou vectorielle) par un champ vectoriel.

TP particules : $p(t+dt) = p(t) + v(t).dt$

- Ce terme est l'advection

$$v(t+dt) = v(t) + \frac{F}{m}.dt$$

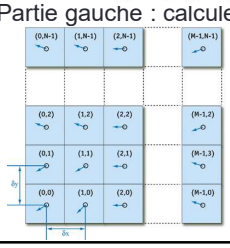
12

## Equation de Saint-Venant : advection

- Equation de Saint-Venant (Shallow Water)

$$\partial\eta/\partial t + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1)$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n\nabla h \quad (2)$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n\nabla h\,. \quad (2')$$

- Partie gauche : calculer par advection

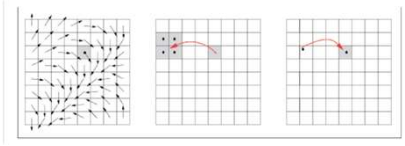


L'advection correspond au transport d'une quantité (scalaire ou vectorielle) par un champ vectoriel.

13

## Equation de Saint-Venant : advection

- Partie gauche : calculer par advection
  - Deux manières de calculer
    - Déplacer la matière vers pos + vitesse
      - Problème d'écriture sur une case « non entière »
    - Déplacer la matière venant de pos - vitesse
      - Suffit d'interpoler la source « non entière » en lecture
      - Plus simple



14

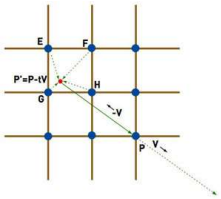
## Equation de Saint-Venant : advection

- Equation de Saint-Venant (Shallow Water)

$$\partial\eta/\partial t + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1)$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n\nabla h \quad (2)$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n\nabla h\,. \quad (2')$$

- Partie gauche : calculer par advection

*Advect*$(s,\mathbf{v})$
(1) **for** $j = 1$ **to** $n_2 - 1$
(2) **for** $i = 1$ **to** $n_1 - 1$
(3) $\mathbf{x} = (i\cdot\Delta x, j\cdot\Delta x)$
(4) $\mathbf{x}' = \mathbf{x} - \Delta t\cdot v(\mathbf{x})$
(5) $s'(i,j) = \text{interpolate}(s,\mathbf{x}')$
(6) **endfor**
(7) **endfor**
(8) **return**$(s')$




15

## Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)

$$\frac{\partial\eta}{\partial t} + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1)$$
$$\frac{\partial\mathbf{v}}{\partial t} + (\nabla\mathbf{v})\mathbf{v} = a_n\nabla h\,, \quad (2)$$

- (1) correspond à la variation de quantité d'eau
  - Variation de la quantité d'eau = dn/dt = -n∇.v = en fonction du champ de vitesse, on fait le bilan des arrivées et des départs en eau
  - Div V : somme du gradient du vecteur vitesse
  - Si plus d'eau part, que d'eau arrive (∇.v>0), ca descend
  - Et inversement.

16

## Eq. Saint-Venant : variation quantité d'eau

- Equation de Saint-Venant (Shallow Water)

$$\partial\eta/\partial t + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1)$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n\nabla h \quad (2)$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n\nabla h\,. \quad (2')$$

- Partie gauche : calculer par advection
- (1) donne variation de n donc n' = n – dt. (n∇.v)

*Update-height*$(\eta,\mathbf{v})$
(1) **for** $j = 1$ **to** $n_2 - 1$
(2) **for** $i = 1$ **to** $n_1 - 1$
(3) $\eta(i,j) -= \eta(i,j)\cdot\left(\frac{(v_1(i+1,j)-v_1(i,j))}{\Delta x} + \frac{(v_2(i,j+1)-v_2(i,j))}{\Delta x}\right)\Delta t$
(5) **endfor**
(6) **endfor**
(7) **return**$(\eta')$

17

## Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)

$$\frac{\partial\eta}{\partial t} + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v} \quad (1)$$
$$\frac{\partial\mathbf{v}}{\partial t} + (\nabla\mathbf{v})\mathbf{v} = a_n\nabla h\,, \quad (2)$$

- (2) correspond à : m.a = m.dv/dt = ∑F
  - Variation de vitesse
  - Ici F=a∇h = gravité dans la direction « vers moins d'eau »
  - le fluide subit une force dans la direction : de plus d'eau vers moins d'eau
  - On change v de tel manière que si bosse l'eau s'étale vers les creux

18

## Eq. de Saint-Venant : variation de v

- (2)(2') donnent variation de v donc
  - $v_1 = v_1 + a_n \nabla h_1$ avec $\nabla h_1 = dh/dx$
  - $v_2 = v_2 + a_n \nabla h_2$ avec $\nabla h_2 = dh/dy$

  Ici $v_1$ correspond à $v_x$ et $v_2$ à $v_y$

*Update-velocities*$(h, v_1, v_2, a)$
(1) **for** $j = 1$ **to** $n_2 - 1$
(2) **for** $i = 2$ **to** $n_1 - 1$
(3) $v_1(i,j) += a(\frac{h(i-1,j)-h(i,j)}{\Delta x})\Delta t$
(5) **endfor**
(6) **endfor**
(7) **for** $j = 2$ **to** $n_2 - 1$
(8) **for** $i = 1$ **to** $n_1 - 1$
(9) $v_2(i,j) += a(\frac{h(i,j-1)-h(i,j)}{\Delta x})\Delta t$
(10) **endfor**
(11) **endfor**

19

## Saint-Venant : intégration

- Shallow-water-step(h,v,g)
  - (1) n = Advect(n,v)
  - (2) v1 = Advect(v1,v)
  - (3) v2 = Advect(v2,v)
  - (4) n' = Update-height(n,v)
  - (5) h = n'+g
  - (6) Update-velocities(h,v1,v2)
  - (7) n=n'

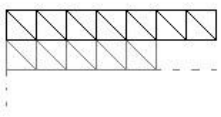
20

## ➔ TP en C/C++/gKit2Light

```
class ShallowWater
{
public:
  ShallowWater();
  void init(const int DIMX, const int DIMY);
  void draw() const;
  void computeOneStep();
protected:
  Array2D m_g;    // height of the ground (0 if flat)
  Array2D m_h;    // height of the water : the thing to compute and to draw
  Array2D m_n;    // m_n = m_h - m_g : amount of water above the ground
  Array2D m_vX;   // velocity along X    (V1 dans ces slides)
  Array2D m_vY;   // velocity along Y    (V2 dans ces slides)
};
```

21

## ➔ TP en C/C++/gKit2Light

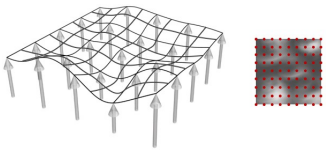


```
class Array2D
{
public:
  const int dimX() const { return m_dimX; }
  const int dimY() const { return m_dimY; }

  float& operator()(const int x, const int y)
  float operator()(const int x, const int y) const
  float interpolate(const float x, const float y)
```

22

## ➔ TP en C/C++/gKit2Light



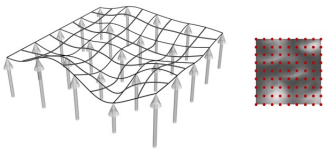
**INIT**

```
m_quad = Mesh(GL_TRIANGLE_STRIP);
m_quad.color( Color(1, 1, 1));

m_quad_texture = read_texture(0, smart_path("data/papillon.png"));
for(i…
        for(j…
                m_quad.normal( 0, 0, 1 );
                m_quad.vertex(  i, j, 0 );          // FLAT
```

23

## ➔ TP en C/C++/gKit2Light



**UPDATE**

```
int id = 0;                 // indice du point dans le tableau
for(i…
        for(j…
                m_quad.vertex(  id, i, j, hm(i,j) );
                id++;
```

24

## Code en 1D ensemble

25

➔ salles de TP …

26

ANCIENNES PARTIES

27

## SHALLOW WATER OPENCL

28

## OpenCL (wikipedia)

- **OpenCL** (**Open** **C**omputing **L**anguage) est la combinaison d'une API et d'un langage de programmation dérivé du C, proposé comme un standard ouvert par le Khronos Group.
- OpenCL est conçu pour programmer des systèmes parallèles hétérogènes comprenant par exemple à la fois un CPU multi-cœur et un GPU.
- OpenCL propose donc un modèle de programmation se situant à l'intersection naissante entre le monde des CPU et des GPU, les premiers étant de plus en plus parallèles, les seconds étant de plus en plus programmables.

29

## OpenCL

- OpenCL can accelerate code by a factor 10 or more
- OpenCL is an open standard
- OpenCL can help save power
- OpenCL can save you hardware cost
- OpenCL adoption is ramping up rapidly
- OpenCL may be used as the basis for generating custom hardware
- The OpenCL C99 language is based on C
- OpenCL can be used from a variety of host languages
- It is easy to start with OpenCL
- OpenCL is platform independent

http://www.amdahlsoftware.com/ten-reasons-why-we-love-opencl-and-why-you-might-too/

30

## OpenGL / OpenCL : le TP

- Le code de départ
  - Carte de hauteur dans une texture 2D (GL)
  - Affichage GL avec un vertex shader
    - Vertex.glsl
  - kernel OpenCL modifie la texture
    - Texture 2D = vu comme une 'image2d'
    - CLWater.cl
    - Kernel = fonction appelée sur chaque case du tableau

31

## OpenCL : un kernel

```
__kernel void shallowWaterInit( const int dimD,
                                __global __write_only image2d_t DD0 )
{
        const int x = get_global_id(0);
        const int y = get_global_id(1);

        if (x>=dimD) return; if (y>=dimD) return;
        const sampler_t sampler =
CLK_NORMALIZED_COORDS_FALSE | CLK_ADDRESS_CLAMP |
CLK_FILTER_NEAREST;

        float4 pixel={0,0,0,0};
        write_imagef( DD0, (int2)(x,y), pixel);
}
```

32

## OpenCL : l'appel

```
clSetKernelArg( m_kernelShallowWater, 0, sizeof(Dim), (void*)&Dim);
clSetKernelArg( m_kernelShallowWater, 1, sizeof(D0), (void*)&D0);

const size_t localWorkSize[] = { LocalWorkSize, LocalWorkSize };
const size_t globalWorkSize[] = { Dim, Dim };

cl_uint workDim = 2;
clEnqueueNDRangeKernel (  m_queue, m_kernelShallowWaterInit,
                          workDim, NULL,
                          globalWorkSize, localWorkSize,
                          0, NULL, NULL);
```

33

## TP ANIMATION PHYSIQUE DE PERSONNAGE AVEC BULLETPHYSICS

34

## TP

- Combiner MoCap et Ragdoll
  - Utiliser Bullet pour la physique
- Regardez le fichier Ragdoll.h/.cpp
  - Ce ragdoll doit avoir la même configuration que le squelette de la MoCap
  - Pour l'instant, c'est un squelette avec 2 membres (bras, avant-bras) qui sont entrés en dur.

35

## TP : Lib BulletPhysics

- Le monde physique gérés par la lib
  - btDynamicsWorld* m_dynamicsWorld;

➔Les objets physiques doivent y être ajoutés

- Dans le TP, il y a une class CAPhysics qui s'occupe du monde physique de Bullet avec
  - computePhysics() : calcul la physique depuis le dernier appel
  - renderPhysics() : affiche les objets physiques, appuyer sur 'P' dans le viewer pour les afficher
  - createRigidBody(…) : ajoute un objet rigide dans le monde physique et renvoie un pointeur dessus

36

## TP : Lib BulletPhysics

- Ragdoll = ensemble de
  - vector<btRigidBody*> m_bodies;
    - Solides rigides
  - vector<btCollisionShape*> m_shapes;
    - Les formes pour les collisions (optionnel)
  - vector<btTypedConstraint* > m_jointsConstraint;
    - Les articulations

37

## TP : Lib BulletPhysics

- Bullet gère les transformations
  - btTransform transform;
    - Rotation + translation
  - Construit avec un quaternion + un vecteur
    - btTransform t( q, v)
  - Construit avec une matrice 3x3 + vecteur*
  - …
  - Toutes les fonctions dont vous avez besoin sont disponibles …

38

## TP : Lib BulletPhysics

- Créer un corps solide btRigidBody
  - Utiliser la fonction de CAPhysics

```
btRigidBody* CAPhysics::createRigidBody(
                float mass,
                const btTransform& startTransform,
                btCollisionShape* shapeForCollision)
```

- Créer une forme pour les collisions

```
new btCapsuleShape( rayon, hauteur );
```

39

## TP : Lib BulletPhysics

- Créer une articulation

```
coneC = new btConeTwistConstraint(A, B, localA, localB);
```

  - A et B sont des btRigidBody
  - localA et localB sont des btTransform relatif à au repère local du btRigidBody
- Articulation avec des contraintes en forme de Cone

- Il existe d'autres types d'articulation
  - btHingeConstraint : mouvement dans le plan type doude
  - …

- Ne pas oublier d'ajouter les articulations au monde physique
  - m_physics.getDynamicsWorld()->addConstraint( coneC, true);
  - true pour ne pas calculer de colision entre 2 btRigidBody relié par l'articulation

40

## TP : Lib BulletPhysics

- Regardez les 2 exemples dans CARagdoll.h

41