Correction TD de Simulation n°2

Cours de simulation et animation

—IMAC troisième année—

Théorie des fluides et algorithme principal

Objectifs

- Equations de fluide incompressible et algorithme de simulation,
- Advection, forces externes et préparation de la projection.

Programme

- Présentation des équations,
- Algorithme de simulation,
- Application des forces externes,
- Préparation de la projection.

▸ Exercice 1. Equations de Navier-Stokes pour un fluide incompressible

Ces équations décrivent le mouvement d'un fluide incompressible. Plus précisément, elles décrivent l'accélération du fluide selon les forces qui s'appliquent dessus. En voici une formulation :

$$\frac{D\vec{u}}{Dt} = \vec{F} - \frac{1}{\rho}\nabla p + \nu\nabla \cdot \nabla\vec{u}$$
$$\nabla \cdot \vec{u} = 0$$

On peut décomposer ces équations en une serie de forces/comportements que le fluides doit respecter, et qui suffisent à décrire son mouvement.

- $\frac{D\vec{u}}{Dt}$: **Accélération** (dérivée matérielle de la vitesse)
- $\vec{F}$: **Forces externes**
- $-\frac{1}{\rho}\nabla p$: **Force inverse au gradient de pression**
- $\nu\nabla \cdot \nabla\vec{u}$: **Force due à la viscosité**
- $\nabla \cdot \vec{u} = 0$: **Incompressibilité**

Nous verrons par l'expérimentation que les erreurs numériques de notre simulation crééent une forme de "flou" dans le mouvement, équivalent visuellement à de la viscosité. Puisqu'il est très difficile de réduire ces erreurs et que l'on souhaite généralement créer des fluide les moins visqueux possibles, on ignore le terme de viscosité.

Les equations sans viscosité sont nommées équations d'Euler :

$$\frac{D\vec{u}}{Dt} = \vec{F} - \frac{1}{\rho}\nabla p$$
$$\nabla \cdot \vec{u} = 0$$

A - Accélération

L'accélération est la variation de la vitesse dans le temps. On s'intéresse ici l'accélération subie par un volume de fluide au passage d'un point de la grille.

Imaginons une quantité q :
$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \vec{u} \cdot \nabla q$ signifie "dérivée matérielle de la quantité q" et correspond selon la formule à la somme de la variation de q dans le temps et de la variation de q dans la direction de déplacement $\vec{u}$.
Autrement dit, le premier terme est la variation de q dans le temps et en un point donné, que l'on corrige en ajoutant, par le second terme, l'influence de la variation de q dans le fluide du au déplacement de la matière sur ce point.
Dit encore autrement, la dérivée matérielle est la variation de la quantité q d'un volume de fluide passant sur un point de la grille.

$\frac{D\vec{u}}{Dt}$ signifie donc la variation de la vitesse d'un volume de fluide passant sur un point de la grille.

On associe souvent au concept de dérivée matérielle le principe de l'**advection**.
L'advection est la manière dont une propriété du fluide est conservée lors de son déplacement (dérivée matérielle nulle).
On peut advecter différentes quantités.
Pour une quantité q, l'advection s'écrit : $\frac{Dq}{Dt} = 0$.
Celà revient à dire que lorsque le fluide se déplace au cours du temps et dans l'espace, la valeur q d'un volume de fluide donné ne change pas.

La couleur est un exemple intuitif d'une donnée du fluide qu'on souhaite voir transportée intacte lorsque le fluide se déplace. Il faut donc advecter la couleur (cette contrainte n'est pas représentée dans les équations ci-dessus).
On souhaitera dans la suite advecter la vélocité ($\frac{D\vec{u}}{Dt} = 0$) , c'est à dire forcer la conservation de la vitesse des éléments du fluide lors de leur déplacement.

B - Forces externes

Dans notre système, les forces externes $\vec{F}$ peuvent être la force exercée par la gravité, le mouvement de la souris pour déplacer artificiellement des zones du fluide...

C - Force inverse au gradient de pression

En un point de l'espace, le gradient de pression ∇p est la direction de plus forte variation de pression. Si une zone à faible pression se situe à gauche d'une zone à forte pression, le gradient de pression est le vecteur allant de gauche à droite.
La force $-\frac{1}{\rho}\nabla p$ est la force qui déplace la matière des zones à forte pression vers les zones à plus faible pression. On voit bien en effet qu'elle grandit dans le sens inverse de celui du gradient de pression (signe "$-$"). Elle est pondérée par l'inverse de la densité du fluide, ce qui signifie qu'un liquide subit moins cette force qu'un gaz.

D - Incompressibilité

$\nabla \cdot \vec{u}$ est la divergence de la vélocité $\vec{u}$, soit la différence entre la quantité de matière entrant dans un volume et celle sortant de ce volume.

Avec un fluide compressible, il serait possible de faire rentrer plus dans un volume que ce qui n'en sortirait (compression), ou bien d'en faire sortir plus qu'il n'en rentrerait (dilatation). Lorsque ces deux quantités sont égales, la divergence est nulle.

Un fluide dont on impose une divergence nulle est un fluide dit "incompressible".
Pour autant que l'on est concerné en synthèse d'images, tous les fluides sont considérés incompressibles (liquides comme gaz), car le taux de compression possible est insignifiant et donc négligeable à l'échelle macroscopique.

▸ **Exercice 2. Simulation numérique**

L'objectif de la simulation est de connaître la nouvelle vélocité en chaque point de la grille, une fois que les différentes forces et contraintes du fluide ont été appliquées pendant un pas de temps Δt.

On procède par la méthode du splitting, qui consiste à découper le problème en étapes de calcul plus simples, réalisées successivement à partir du résultat du calcul précedent. Cette méthode permet notamment d'organiser la séquence des calculs de sorte que les propriétés du champ de vitesse en sortie de chaque calcul soient bien celles nécessaires en entrée des étapes suivantes.

Par exemple, la phase de calcul de l'advection de la vitesse ne peut être justement réalisée que sur un champ de vitesses non divergent. On s'assure donc lors de la mise en place du splitting que le calcul précedant l'advection respecte bien la contrainte de non-divergence.

Voici la séquence des opérations à réaliser :

- Initialisation d'un champ de vélocités non-divergent
- Pour les pas de temps $n = 0, 1, 2 \ldots$
  - $\Delta t \leftarrow$ Détermination de la durée optimale entre t_n et t_{n+1}
  - $\vec{u}^A \leftarrow$ Advection de $\vec{u}^n$ par $\vec{u}^n$ pendant Δt
  - $\vec{u}^B \leftarrow \vec{u}^A + \Delta t \vec{F}$
  - $\vec{u}^{n+1} \leftarrow$ "Projection" : Correction de la vitesse $\vec{u}^B$ de sorte de respecter à la fois l'incompressibilité et les conditions aux bords, pendant Δt

Préparez une fonction `void Simulation::update()` dans laquelle ces étapes seront appellées (sauf le calcul deΔt, ajoutez par contre l'advection de la couleur).

Appellez update et render non pas dans `void Application::animate()` mais dans `void Application::loop()`, juste avant `renderFrame();`. Ainsi, les calculs ne seront plus constraints à un timer mais auront lieu pendant une durée extensible à chaque frame.

✂ -

```
// Simulation update
void Simulation::update()
{
    GLfloat dt=1.0/30.0;

    // Colors semi-lagrangian advection (and potential visualization update)
    //this->advectColors(dt);

    // Velocities semi-lagrangian advection (and potential visualization update)
    //this->advectVelocities(dt);

    // Extern forces are applied
    //this->applyForces(dt);

    // Velocities are uptated to satisfy incompressiblity and limit conditons
    //this->project(dt);
}
```

- ✂

A - Détermination de Δt

Dans notre simulation, nous fixons le pas de temps.

Il devrait en théorie être calculé selon la magnitude des forces appliquées sur la matière et la largeur des cellules de la grille. Plus la grille est fine, plus le pas de temps doit être court : l'idée est de limiter les déplacements à chaque pas pour qu'ils soient inférieurs à la taille des cellules Δx.
La formule suivante est appropriée : $\Delta t = k\frac{\Delta x}{\vec{u}_{max}}$.

Une bonne heuristique pour la vélocité maximale $\vec{u}_{max}$ est : $\vec{u}_{max} = max(|\vec{u}|) + \sqrt{\Delta x|\vec{F}|}$

En théorie, le Δt doit être évalué pour chaque n selon l'état de la simulation. En attendant d'en arriver à un avancement ultérieur du programme, on peut fixer le pas de temps de manière approximative, en gardant en tête les principes illustrés par cette formule.

B - Advection

Dans la fonction update, décommentez les fonctions d'advection (couleur et vélocité). Elles sont en théorie déjà réalisées depuis le TD précédent.

C - Forces exterieures

a) Construisez un tableau `forces` de $nbSamples * 4$ cases, membre de la classe Simulation. Il contiendra les forces externes au centre de chaque cellule de la grille.

```
// in Simulation.hpp
        GLfloat * forces; // Samples forces

        void buildForces(Object * object);
        void drawForces();

// in Simulation::Simulation(...)
        // Samples forces
    this->forces=new GLfloat[this->nbSamples*4];
```

b) Faites-en l'initialisation dans void Simulation::initSamples(). Vous pouvez par exemple mettre les forces à nul partout sauf dans une colonne centrale de forces orientées vers y.

```
// in Simulation::initSamples()

    forces[index*4+0]=0.0;
    forces[index*4+1]=0.0;
    forces[index*4+2]=0.0;
    forces[index*4+3]=0.0;
    if (iX==(nbSamplesX)/2) forces[index*4+1]=1.0;
```

c) Construisez les fonctions `void Simulation::buildForces()` et `void Simulation::drawForces()` appropriées pour le dessin (dessin sous forme de lines pour les vecteurs des forces).

```
// Builds an object to visualize the forces
void Simulation::buildForces(Object * object)
{
    std::cout<<"Building_on_GPU_:_cell_forces"<<std::endl;

    object->nbVertices=nbSamples*2;
    object->nbIndices=object->nbVertices;

    GLfloat vertices[object->nbVertices*4];
    GLfloat colors[object->nbVertices*4];
    GLuint indices[object->nbIndices];

    for (GLuint iY=0 ; iY<nbSamplesY ; iY++)
    {
```

```
        for (GLuint iX=0 ; iX<nbSamplesX ; iX++)
        {
            GLuint iSamples=iY*nbSamplesX+iX;
            GLuint index=iSamples*2+0;
            // Position of vectors beginning
            vertices[index*4+0]=samples[iSamples*4+0];
            vertices[index*4+1]=samples[iSamples*4+1];
            vertices[index*4+2]=samples[iSamples*4+2];
            vertices[index*4+3]=1.0;
            colors[index*4+0]=1.0;
            colors[index*4+1]=1.0;
            colors[index*4+2]=1.0;
            colors[index*4+3]=1.0;
            indices[index]=index;

            index=iSamples*2+1;

            // The vector is printed from white to red
            vertices[index*4+0]=samples[iSamples*4+0]+forces[iSamples*4+0]*vectorScale;
            vertices[index*4+1]=samples[iSamples*4+1]+forces[iSamples*4+1]*vectorScale;
            vertices[index*4+2]=samples[iSamples*4+2]+forces[iSamples*4+2]*vectorScale;
            vertices[index*4+3]=1.0;
            colors[index*4+0]=1.0;
            colors[index*4+1]=0.0;
            colors[index*4+2]=0.0;
            colors[index*4+3]=1.0;
            indices[index]=index;
        }
    }

    // Sends the data into buffers on the GPU
    object->sendPrimitives(vertices, indices);
    object->sendColors(colors);
}

// Creates, builds and add to draw list an object for forces visualization
void Simulation::drawForces()
{
    Object * objectForces=new Object(GL_LINES);
    GLuint storedObjectForces=scene->storeObject(objectForces);
    this->buildForces(objectForces);
    GLuint forcesID=scene->addObjectToDraw(storedObjectForces);
    scene->setDrawnObjectShaderID(forcesID, defaultShaderID);
}
```

d) Verifiez votre tableau gràce à l'affichage.

e) Réalisez la fonction `void Simulation::applyForces(GLfloat dt)` qui modifie le champ de vélocité selon Δt et les forces sur la grille. Utilisez par exemple la formule mentioné dans l'algortihme de simulation ci-dessus (de type forward Euler).

f) Appellez la fonction et vérifiez son influence. On constate que le fluide vient s'accumuler là où les forces s'arrêtent, créant des zones plus "denses", et des zones vides. La condition d'incompressibilité n'est pas respectée. La partie la plus délicate de la simulation consiste à corriger les vitesses de sorte de respecter cette contrainte.

D - Projection

Cette dernière partie a pour but de corriger les vitesses pour respecter l'incompressiblité tout en renforçant les conditions de vitesses aux bords.
Cette étape est difficile car elle implique la recherche d'une solution optimale sur l'ensemble des cellules. Autrement dit, le champ de vitesses résultat implique la résolution d'un système linéaire.

Pour être plus précis, on peut trouver les nouvelles vitesses facilement, à partir du moment où les pressions en chaque cellule de la grille sont calculées. C'est ce calcul des pressions qui requiera la résolution du système.

a) Créez et appellez la fonction `void Simulation::project()`.

b) Initilisez votre tableau `pressures` dans initSamples().

```
// in Simulation::initSamples()

    pressures[index]=1.0;
```

c) Faites les fonctions `void Simulation::buildPressures()` et `void Simulation::drawPressures()`. Cette fois, vous devez également ajouter un `Object * objectPressures` dans Simulation.hpp et prévoir la mise à jour du VBO à chaque appel de la fonction de projection, de sorte de pouvoir contrôler visuellement l'évolution de la pression (basez vous sur la fin de `void Simulation::advectColors(GLfloat dt)`). N'oubliez pas d'initiliser l'`Object *` à `NULL` pour pouvoir tester son existence dans `project()` avant d'updater le VBO (si les pression ne sont pas dessinées, ça évitera la seg fault).

```
// in Simulation.hpp
        Object * objectPressures; // For rendering pressures (pressure checking)

        void buildPressures(Object * object);
        void drawPressures();

// in Simulation::Simulation(...)
        // For rendering pressures (pressure checking)
    this->objectPressures=NULL;

// Builds an object to visualize the pressures
// The object will then be accessed and updated each frame (project(...))
// 3D working
void Simulation::buildPressures(Object * object)
{
    std::cout<<"Building_on_GPU_:_samples_positions"<<std::endl;

    object->nbVertices=nbSamples;
    // No indices are necessary since we draw GL_POINTS
  GLuint * indices=NULL;

    GLfloat colors[object->nbVertices*4];
    for (GLuint iSamples=0 ; iSamples<nbSamples ; iSamples++)
    {
            colors[iSamples*4+0]=pressures[iSamples];
            colors[iSamples*4+1]=pressures[iSamples];
            colors[iSamples*4+2]=pressures[iSamples];
            colors[iSamples*4+3]=1.0;
    }

    // Sends the data into buffers on the GPU
    object->sendPrimitives(samples, indices);
    object->sendColors(colors);

    // The colors buffer will be updated each frame so is stored in dynamic memory
    glBindBuffer(GL_ARRAY_BUFFER, objectPressures->colorsVboId);
    glBufferData(GL_ARRAY_BUFFER, objectPressures->nbVertices*4*sizeof(GLfloat), colors,
GL_DYNAMIC_DRAW);
}

// Creates, builds and add to draw list an object for pressures visualization
void Simulation::drawPressures()
{
    this->objectPressures=new Object(GL_POINTS);
    GLuint storedObjectPressures=scene->storeObject(objectPressures);
    this->buildPressures(objectPressures);
    GLuint pressuresID=scene->addObjectToDraw(storedObjectPressures);
    scene->setDrawnObjectShaderID(pressuresID, defaultShaderID);
}
```

d) Réalisez la partie de la fonction `project()` dans laquelle vous utilisez les valeurs de pression courante (p) pour mettre à jour les vélocités. Les formules sont les suivantes :

$$velocitiesX_{iX+\frac{1}{2}} -= \Delta t \frac{1}{\rho} \frac{p_{iX+1} - p_{iX}}{\Delta x}$$

$$velocitiesY_{iY+\frac{1}{2}} -= \Delta t \frac{1}{\rho} \frac{p_{iY+1} - p_{iY}}{\Delta x}$$

La grille MAC se justifie pour ces formules. Elle permet en effet que les valeurs de velocités soient exactement entre deux points de pression connue, ce qui facilite le calcul de ∇p sur les "borders" et donc l'utilisation de ce gradient dans l'update des velocités.

e) Faites une fonction de calcul de la divergence.
Utilisez le protocole `void Simulation::setDivergences(double * divergences)` qui parcourt les cellules du tableau dont l'adresse est en paramètre et associe à chaque case la divergence correspondante.
Utilisez la formule suivante :

$$(\nabla \cdot \vec{u})_{(iX,iY)} = \frac{velocitiesX_{iX+\frac{1}{2}} - velocitiesX_{iX-\frac{1}{2}}}{\Delta x} + \frac{velocitiesY_{iY+\frac{1}{2}} - velocitiesY_{iY-\frac{1}{2}}}{\Delta x}$$

Pour terminer la fonction `project()`, il faudra résoudre un système de type $Ap = b$ avec A une matrice représentant les influences entre cellules voisines, p la pression recherchée et b un vecteur proche de celui des divergences. Nous construirons ces éléments en nous basant sur la théorie vue plus haut et résoudrons le système dans un TD ultérieur.

▸ **Exercice 3. En préparation du projet**

A - De nouvelles idées pour la forme ?

Les vitesses, envisagées dans un premier temps pour la construction de la forme, sont modifiées par l'algorithme au cours de la simulation.

Les forces ont aussi une influence sur les motifs créés mais offrent une alternative interessante (car stable dans le temps) pour le dessin de formes (forces dans le squelette ou aux bords d'une forme). Que pensez-vous de cette solution ?

On peut également envisager que les formes soient par leur mouvement, génératrices de forces. L'ensemble des deux sens de l'interaction force/forme peut peut-être aussi donner un résultat interessant.

B - Inspirations visuelles

Cherchez des vidéos de "cloud tanks" et d'insertion d'encre dans de l'eau. Ces exemples peuvent, au même titre que ceux visualisés au premier TD, vous inspirer dans votre recherche esthétique.