Compte Rendu - Projet Modélisation et Programmation C++

BAYET Théophile, MOUNE Paul

Mai 2017

Table des matières

1 Introduction

Ce document est le compte-rendu du projet de modélisation et programmation C++. Il regroupe les réponses aux questions des différents TP ainsi que nos commentaires sur notre implémentation et diverses remarques.
L'objectif de ce projet est de modéliser et visualiser un océan.

2 TP1

L'objectif de ce TP est d'implémenter une classe Dvector qui sera ensuite utiliser dans notre modèle pour stocker les différentes hauteurs des points.

Nous avons donc créer plusieurs constructeurs ainsi que le destructeur. On retrouve le constructeur par défaut mais aussi un constructeur par recopie et un constructeur par lecture de fichier.

Nous avons aussi implémenter une méthode *display* qui permet d'afficher sur la sortie standard l'objet de la classe Dvector.

2.1 Questions :

Considérons le code :

$Dvector\ x;$
$x = Dvector(3, 1.);$

Ici, on construit d'abord un élément de la classe Dvector nommé x sans l'initialiser. On utilise ensuite l'opérateur = qui a été redéfini pour la classe Dvector et qui va assigner à x une valeur.

En revanche, si l'on écrit directement :

$Dvectorx = Dvector(3, 1.);$

On construit directement le Dvector x avec une valeur en utilisant le constructeur adéquat que l'on a défini.

3 TP2

Le but de ce TP est d'enrichir notre classe Dvector créee lors du premier TP en implémentant différentes méthodes.

D'une part, il s'agit d'implémenter les opérateurs classiques pour qu'ils fonctionnent sur notre classe, comme par exemple l'addition avec un autre Dvector ou encore la multiplication par un scalaire.

Nous créons aussi un opérateur d'accession à un élément du Dvector. De plus, nous surchargeons les opérateurs d'affectation et de test d'égalité.

4 TP3

Dans ce TP, nous allons déclarer plusieurs classes qui vont être utilisées pour modéliser l'océan.

Dans un premier temps, il nous faut une classe $Height$ qui va utiliser la classe Dvector que nous avons implémentée avant. Cette classe contiendra l'ensemble des valeurs des hauteurs claculées dans la "boîte" qui servira d'océan.

Ainsi, la classe Height est composée de 2 entiers, les dimensions de la boîte, et d'un Dvector de taille $DimX * DimY$.

Nous créons les différents constructeurs qui vont nous être utiles ainsi que le destructeur.

Les différentes méthodes implémentées pour cette classe sont les accesseurs aux dimensions et qu'un opérateur () permettant d'accéder à la valeur de la hauteur stockée en (i, j).

De plus, nous avons besoin d'une méthode *set* qui modifie la valeur de la hauteur en un endroit particulier.

Dans un second temps, nous allons définir une classe mère, $WaveModel$, qui va nous servir pour implémenter l'effet des vagues dans l'océan. Plus tard, plusieurs classes hériterons de celle-ci.

Il ne faut pas que les différents paramètres de cette classe soient modifiables, ils sont donc privées et nous ne définissons pas de fonctions pour *set*.

En revanche, nous devons définir des accesseurs ainsi que plusieurs types de constructeurs.

De plus, nous implémentons un opérateur () qui permet de considérer cette classe comme un foncteur.

Enfin, nous implémentons une première modélisation de vague avec la classe $GernsterWaveModel$ qui dérive donc de la classe $WaveModel$.

4.1 Questions :

Nous avons donc, dans notre implémentation de la modélisation de l'océan, créer plusieurs classes. Comme il existe plusieurs façons de représenter les vagues,

la classe $WaveModel$ va être dérivée pour créer plusieurs modèles.

D'une part, on aura la classe $GernsterWaveModel$, qui héritera donc de la classe $WaveModel$. D'autre part, on aura aussi la classe $PhilipsWaveModel$, qui sera décrite dans la prochaine section, qui héritera de la même manière de la classe $WaveModel$.

De plus, la classe $Height$ qui va être utilisée dans nos modèles de vagues va contenir un tableau à 2 dimensions qui sera implémenté grâce à la classe $Dvector$ que nous avons créée au début.

Pour gérer l'apparence des vagues modélisées dans notre océan, plusieurs paramètres vont influencer la violence de la houle.

Ainsi, dans la classe $WaveModel$, nous pourrons contrôler la houle grâce aux paramètres d'intensité et de hauteur maximum de vague.

5 TP4

L'objectif de ce TP est de trouver et implémenter un nouveau modèle de visualisation de la houle dans l'océan.

Pour cela, nous allons utiliser une représentation spectrale du champ de hauteur. Nous créons donc une classe $PhilipsWaveModel$ qui dérive directement de la classe $WaveModel$ créer auparavant.

Pour cette représentation, il faut implémenter la transformée de Fourier ainsi que son inverse et l'appliquer sur notre champs de hauteurs.

On implémente donc la Fast Fourier Transform, FFT, avec l'algorithme de Cooley-Tukey. Pour pouvoir l'appliquer sur notre champ de hauteurs, il faut modifier la classe $Dvector$ que nous avons implémenté pour qu'elle gère une partie imaginaire.

Maintenant que la classe est créée et l'implémentation de la FFT faite, il nous faut maitenant créer la classe Océan qui va nous servir pour regrouper tout le travail fait et permettre la visualisation de notre modèle.

On lui donne en paramètres les différentes dimensions nécessaires à la modélisation et à la visualisation, ainsi que le temps, le modèle de houle utilisé avec un paramètre de la classe $WaveModel$ et un Dvector qui contient les hauteurs de la houle.

6 Tests

Toutes les classes et leurs méthodes qui ont été présentées ici ont été testées par plusieurs tests pertinents afin de veiller à ce que notre implémentation soit correcte.

Ces tests vont en effet tester pour toutes les classes implémentées les constructeurs, les operateurs et les méthodes propres à chaque classe.

Nous avons dans un premier temps un fichier de tests pour la classe *Dvector*.

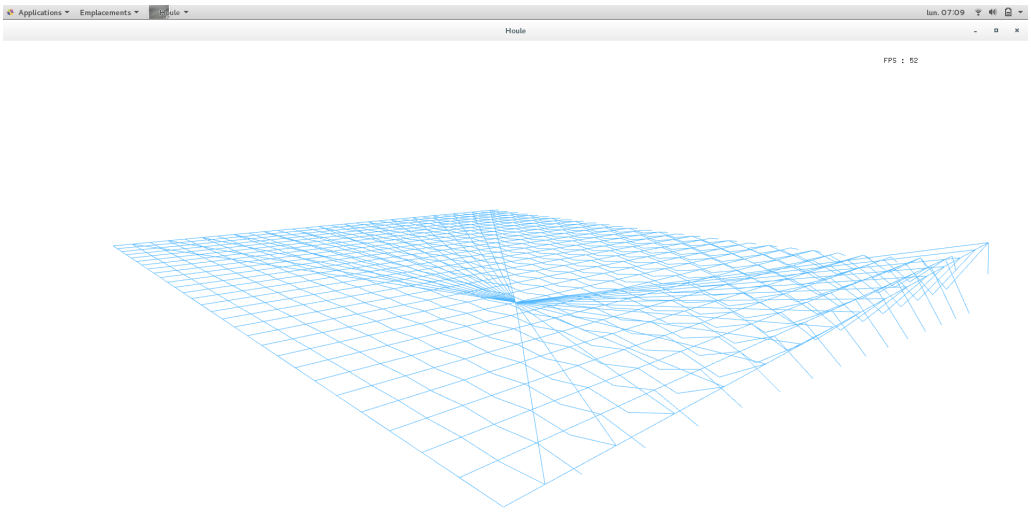
```
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from DvectorTest
[ RUN      ] DvectorTest.constructors
[       OK ] DvectorTest.constructors (0 ms)
[ RUN      ] DvectorTest.internalFunctions
[       OK ] DvectorTest.internalFunctions (0 ms)
[ RUN      ] DvectorTest.operators
[       OK ] DvectorTest.operators (0 ms)
[----------] 3 tests from DvectorTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 3 tests.
```

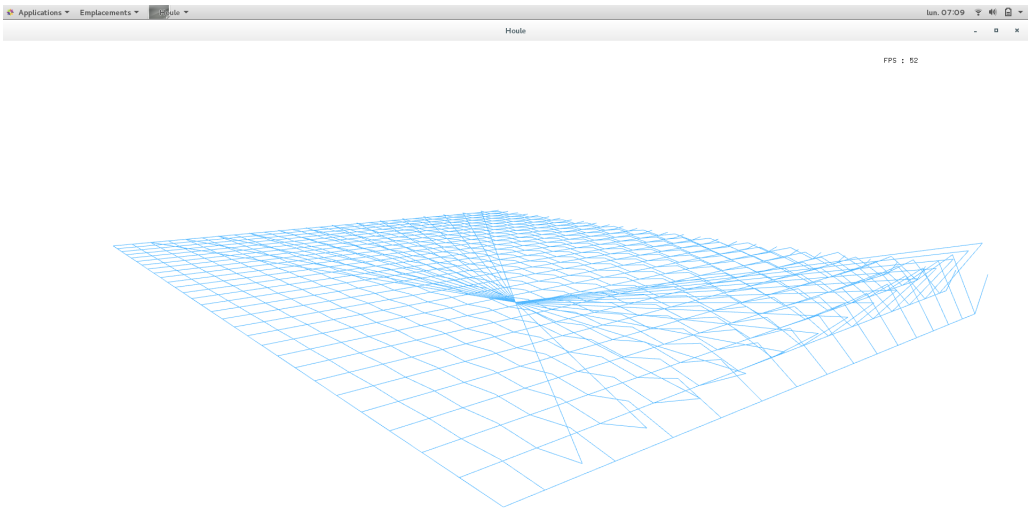
De même, nous avons un fichier de tests pour les classes Height, Ocean et les différentes implémentations des vagues.

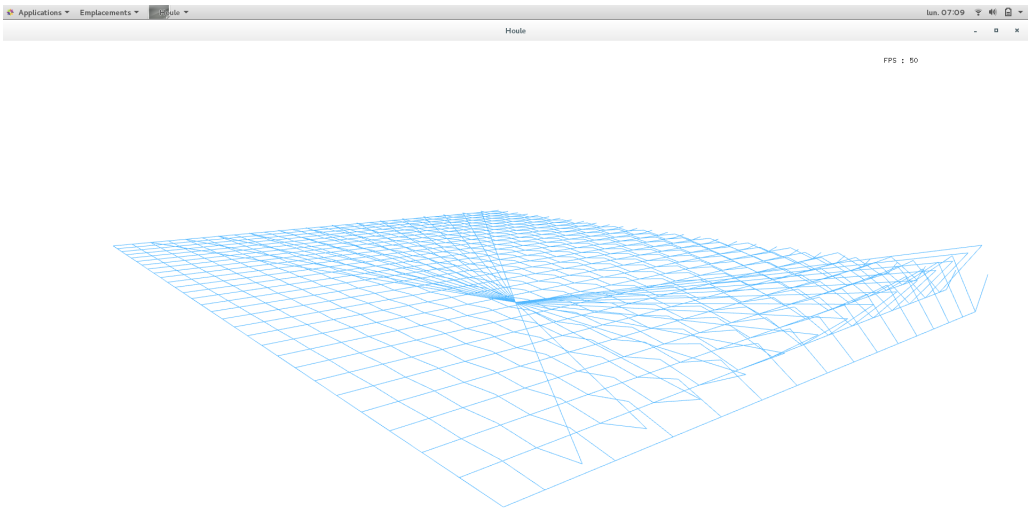
```
[==========] Running 5 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 5 tests from WavesTest
[ RUN      ] WavesTest.constructorsAndAccessors_Height
[       OK ] WavesTest.constructorsAndAccessors_Height (0 ms)
[ RUN      ] WavesTest.constructorsAndAccessors_WaveModel
[       OK ] WavesTest.constructorsAndAccessors_WaveModel (0 ms)
[ RUN      ] WavesTest.constructorsAndAccessors_GerstnerWave
[       OK ] WavesTest.constructorsAndAccessors_GerstnerWave (0 ms)
[ RUN      ] WavesTest.constructors_PhilipsWaveModel
[       OK ] WavesTest.constructors_PhilipsWaveModel (0 ms)
[ RUN      ] WavesTest.constructorsAndAccessors_Ocean
[       OK ] WavesTest.constructorsAndAccessors_Ocean (0 ms)
[----------] 5 tests from WavesTest (0 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 5 tests.
```

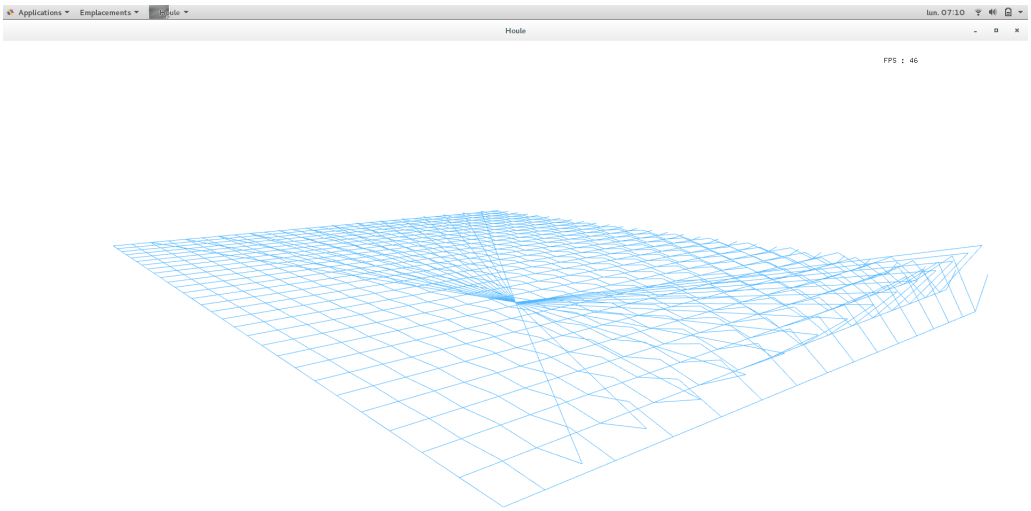
7 Visualisation et résultats

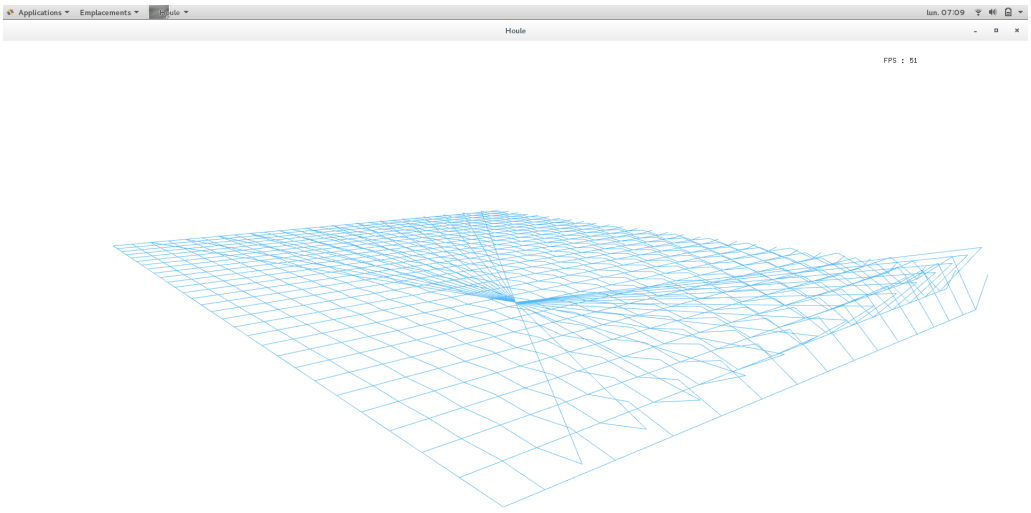
Voici en image les résultats que nous avons obtenus :

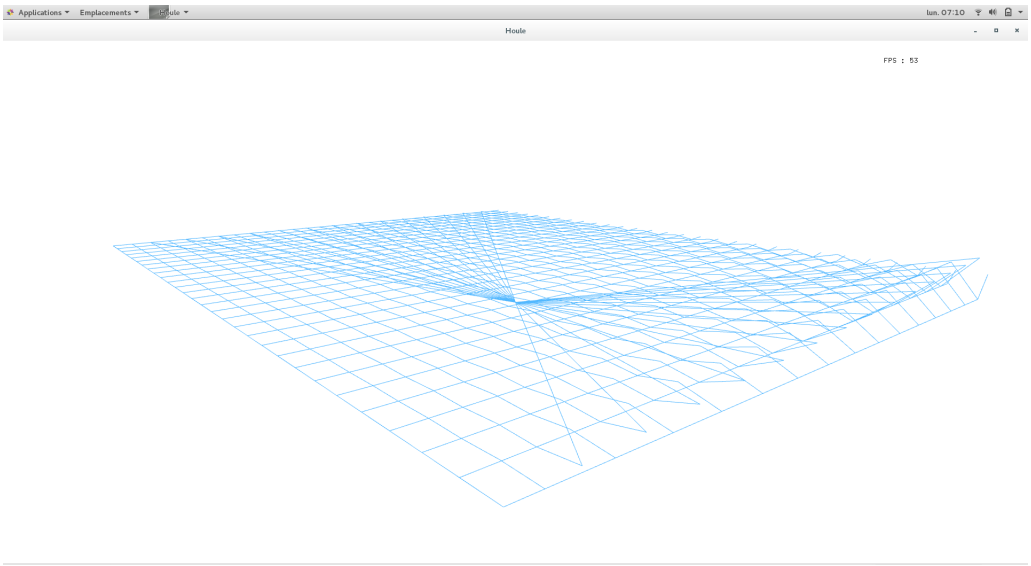


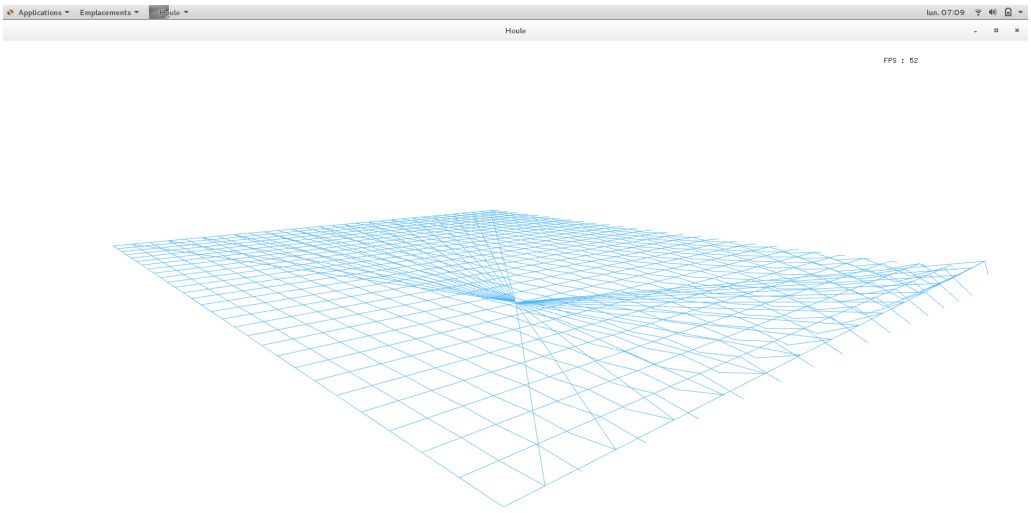

Applications
Emplacements
Houle
lun. 07:09
Houle
FPS : 52

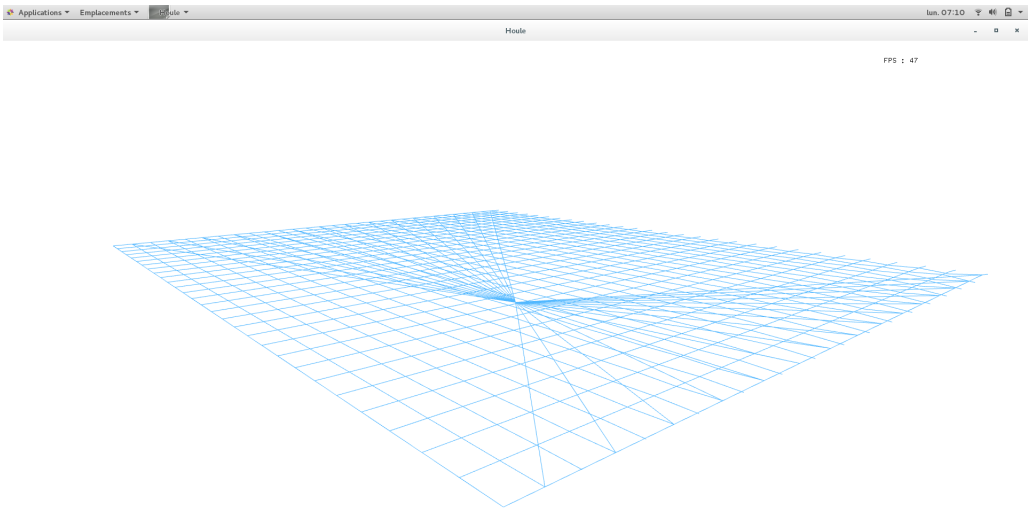

Applications
Emplacements
Houle
lun. 07:09
Houle
FPS : 50


Applications
Emplacements
Houle
lun. 07:10
Houle
FPS : 46


Applications
Emplacements
Houle
lun. 07:09
Houle
FPS : 51


Applications
Emplacements
Houle
lun. 07:10
Houle
FPS : 53


Applications
Emplacements
Houle
lun. 07:09
Houle
FPS : 52



8 Conclusion

En conclusion, notre modèle de vague n'est vraiment pas parfait. En revanche, on aperçoit tout de même une vague sur le côté droit.

On remarque que cette vague s'amortit bien au fur et à mesure jusqu'à disparaître à la fin de l'animation.

On peut dire que, modulo les problèmes d'affichage, notre modèle semble fonctionner pour modéliser une vague.

Avec plus de temps, il faudrait que nous reglions ces problèmes d'affichage afin de voir apparaître des vagues partout sur notre océan.