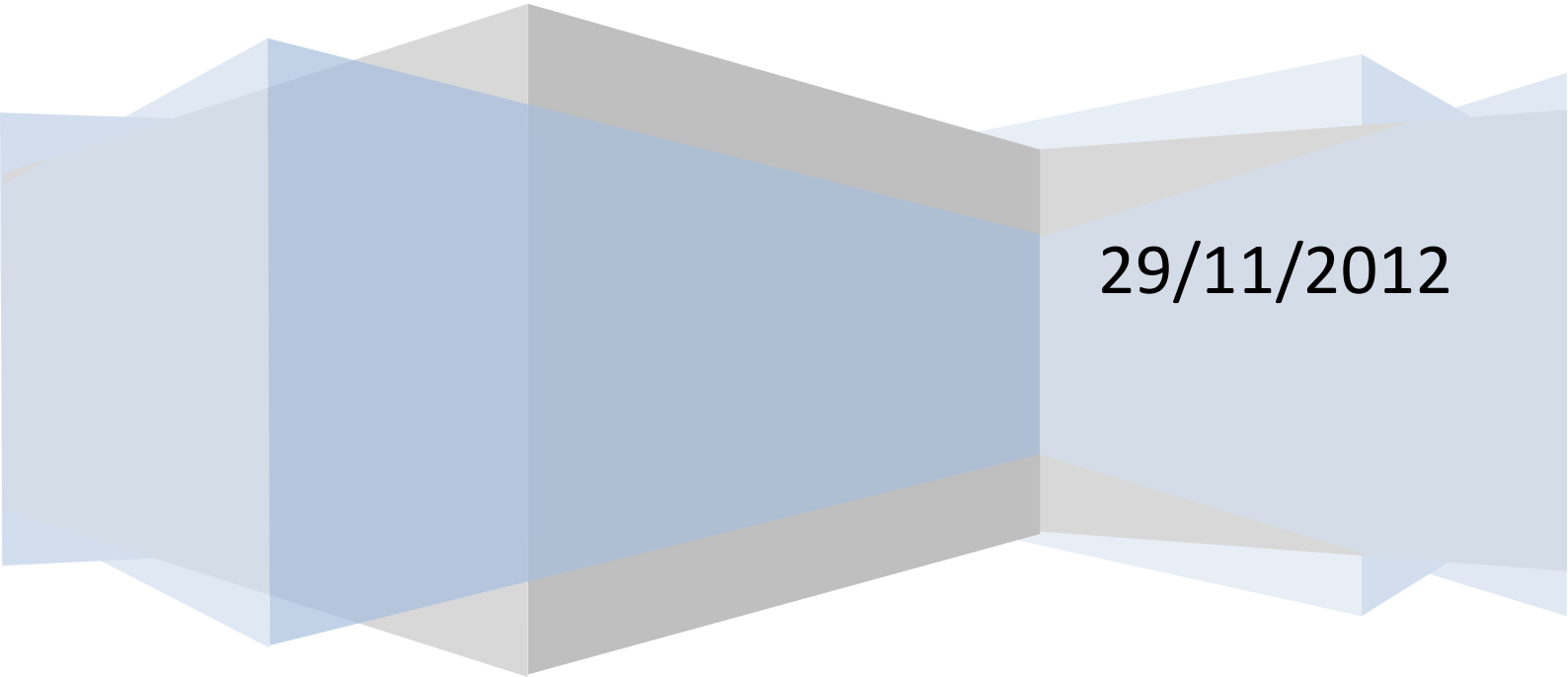


# NeoAxis engine

Champ de vecteurs

MONIER Vincent



29/11/2012

## Sommaire

---

|  |   |
|--|---|
| NeoAxis: Affichage d'un champ de vecteurs..... | 2 |
| Description.....                               | 2 |
| Réalisation.....                               | 3 |
| Suite du travail.....                          | 7 |

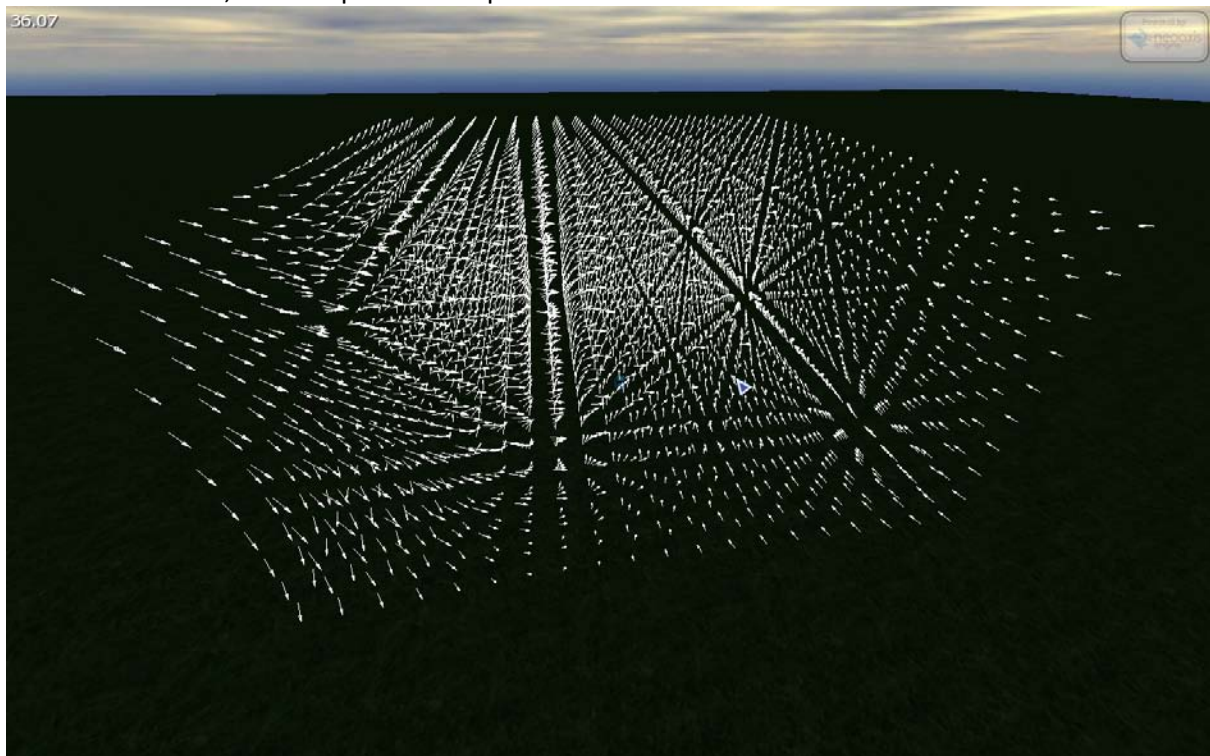
## NeoAxis: Affichage d'un champ de vecteurs

### Description

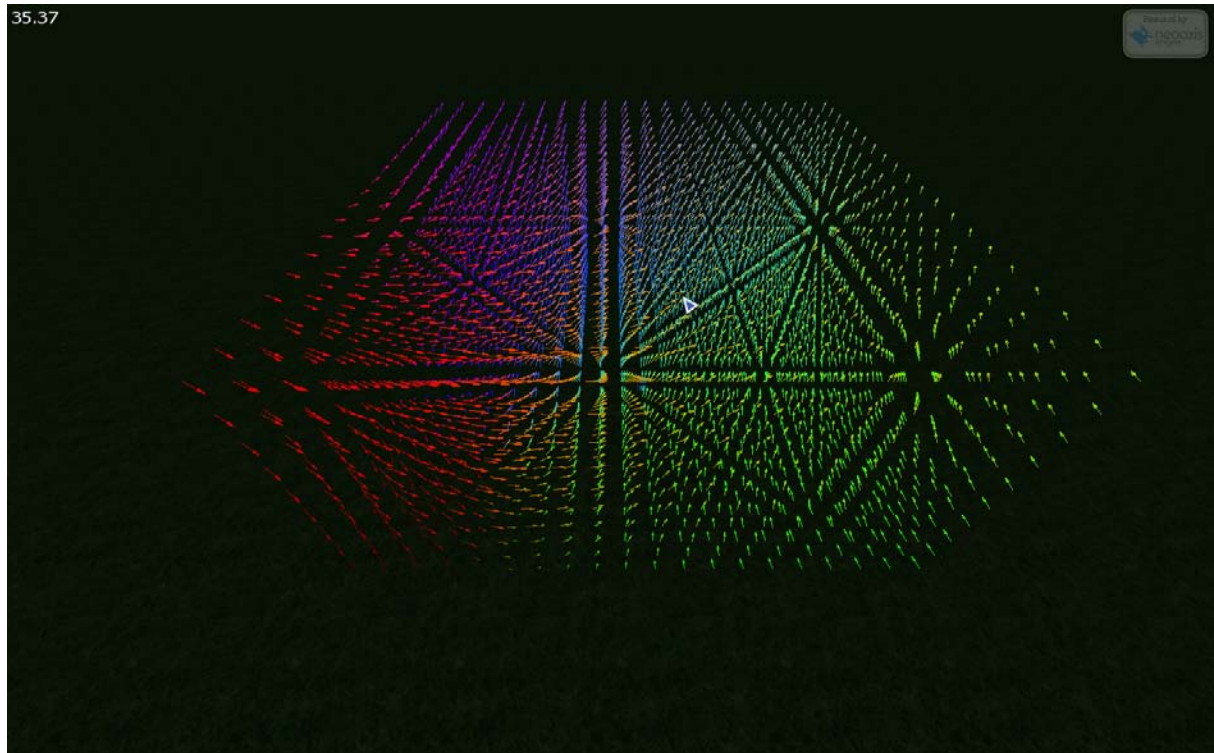
Certaines données géographiques peuvent se présenter sous la forme d'un champ de vecteurs 3D, dans un espace 3D, par exemple, la force et la direction du vent. Certaines données en dimensions inférieures pourront se représenter de la même façon (par exemple, une carte 2D des vents peut se représenter dans l'espace 3D).

Il est donc simple d'afficher une donnée vectorielle de dimension inférieure ou égale à 3, mais dans le cas d'une dimension supérieure, cela peut être un peu plus délicat. Pour des dimensions 4, 5 et 6, il est conseillé d'utiliser la couleur : le rouge représentera la dimension 4, le vert la 5 et le bleu la 6<sup>e</sup>. En revanche, il ne sera pas possible de représenter des données de dimension supérieure à 6. De plus, si l'espace 3D est « faiblement » borné (on peut aller de -10.000 à +10.000), l'espace des couleurs est nettement plus réduit, puisque chaque composante s'étale de 0 à 255 inclus (en pratique, elles pourront s'étaler de 0.0 à 1.0 inclus, mais la précision de l'affichage ne sera alors pas suffisante pour afficher des données précises via les couleurs).

Ci-dessous, un exemple de champ de vecteurs :



*Exemple de champ de vecteur, sans couleur*

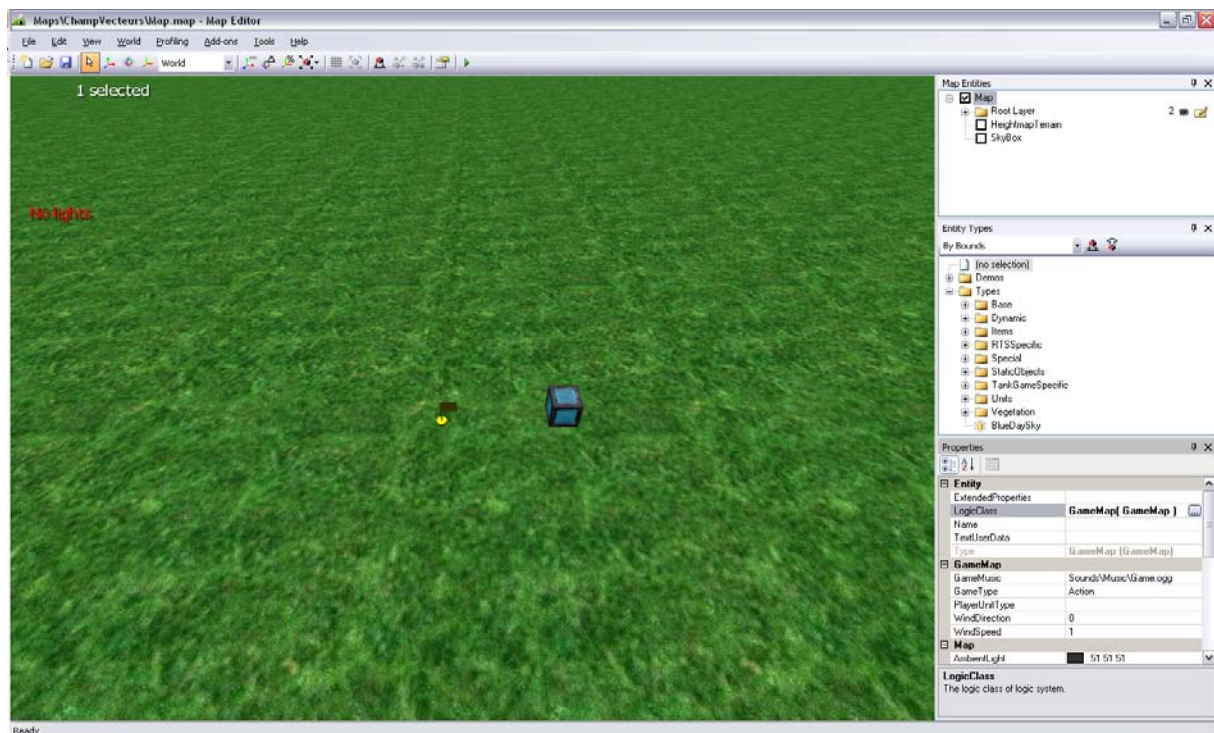


*Avec la couleur en plus pour afficher des données en 6 dimensions (XYZ+RGB)*

## Réalisation

Pour réaliser cet affichage, il faudra éditer le script C# de la carte de l'environnement 3D. Pour cela, ouvrez la carte, sélectionnez « Map » dans « Entities », et double-cliquez, dans « propriétés », sur « LogicClass » jusqu'à ce que la carte soit associée à la classe logique « GameMap ».

Créez, sur cette classe « GameMap », la méthode « Render » :



*L'objet « Map » est associé à la LogicClass « GameMap ».*



Créez la méthode « Render » pour la classe « GameMap ».

Dans cette méthode, insérez le code suivant :

```

Camera c = camera;
if (c != RendererWorld.Instance.DefaultCamera)
    return; //Vérifie que la caméra est la caméra d'affichage à l'écran et non celle d'un reflet

//-----
// Paramètres modifiables
//-----
float gridsize = 40f; //Taille du cube de vecteurs à afficher
int n = 20; // Nombre de vecteurs dans ce cube à afficher, selon chacun des axes
Vec3 o = new Vec3(13f, 11f, 1f); // Position du centre du cube
//Vec3 o = c.Position; //Cette position peut être liée à celle de la caméra,
// pour que le centre du cube soit toujours coïncidant avec cette caméra

//-----
// Code d'affichage
//-----
Vec3 omin = o - new Vec3(1, 1, 1) * 0.5f * gridsize; // Position du point bas
float odelta = gridsize / n; // Ecart entre deux vecteurs le long de l'axe X, ou Y ou Z

for (int i = 0; i < n; i++) //pour chaque rangée
{
    for (int j = 0; j < n; j++) //pour chaque ligne
    {
        for (int k = 0; k < n; k++) // pour chaque colonne
        {
            Vec3 p = omin + new Vec3(i, j, k) * odelta; // Calculer la position du point
            Vec3 x = PointManager.EvalChamp( p); // Calculer la valeur en ce point
            Vec3 w = x; // Alias
            w.Normalize(); // Alias, ici, on utilise la valeur XYZ pour définir la couleur
        }
    }
}

```

```

        c.DebugGeometry.Color = new ColorValue(w.X, w.Y, w.Z);
        // Définit la couleur du vecteur, format Red Green Blue de 0 à 1
        c.DebugGeometry.AddArrow(p, p+x, 1f, false); // Affiche un vecteur
    }
}

```

Ce code va donc afficher un champ de vecteurs selon une grille dont la taille totale (gridsize) et la résolution (n) sont fixées. En chaque point de la grille, la valeur du champ est évaluée (valeur de x, ici calculée par `PointManager.EvalChamp(p)` où p est la position du point dont on cherche le champ). La couleur du vecteur est alors définie (attribut public « Color » de « Camera.DebugGeometry »). Le vecteur peut alors être ajouté (il sera de la couleur définie par « Camera.DebugGeometry.Color ») via la méthode « Camera.DebugGeometry.AddArrow », qui prend en argument le point de départ du vecteur, le point d'arrivée, la taille de la flèche en décimal « float », et un booléen indiquant s'il faut masquer le corps de la flèche (true) ou non (false) :



*Le corps de flèche est la partie en rouge : cette partie sera masquée si le dernier argument de « AddArrow() » est à « true ».*

La documentation présente d'autres méthodes de « Camera.DebugGeometry » permettant d'ajouter des traits, des sphères, des flèches, des triangles, ... avec un aspect filiforme (vue « wireframe »). D'ailleurs, « DebugGeometry » pourrait être utilisée pour afficher des textes (labels) simples dans l'environnement 3D (un peu à l'image des anciens afficheurs digitaux, qui affichent les lettres grâce à des « bâtons » :

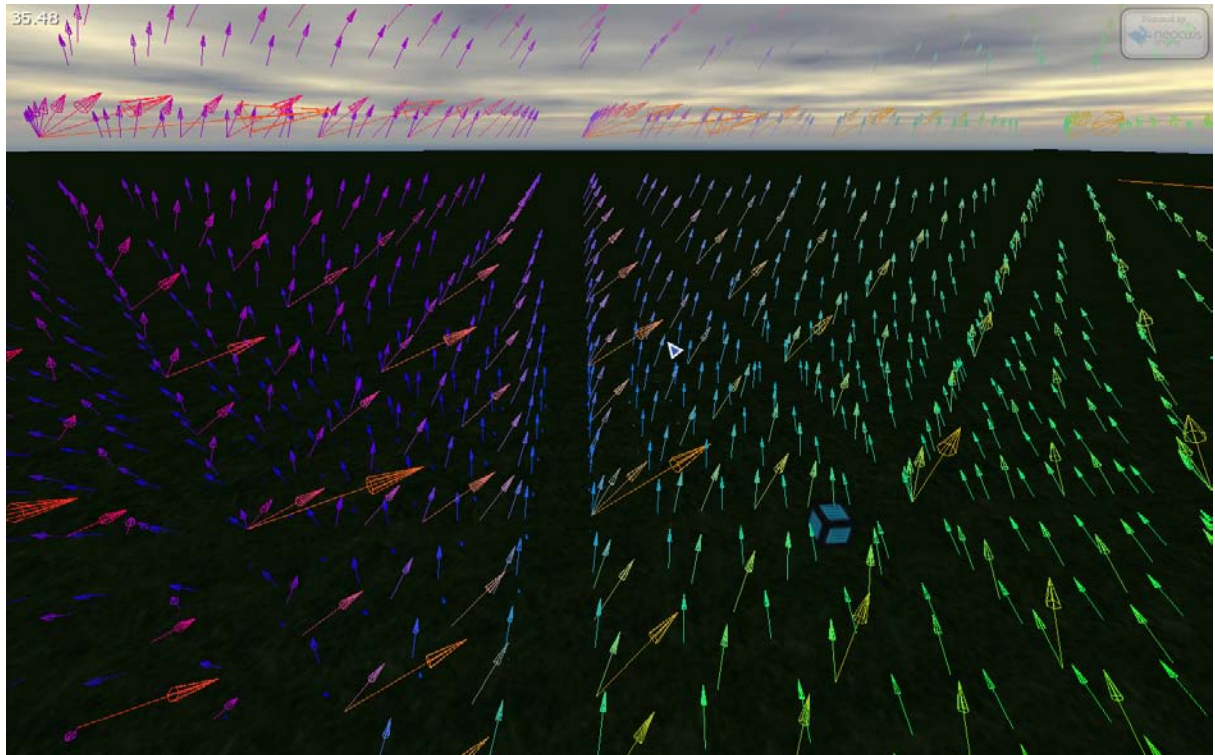


*Le système des afficheurs numériques pourrait être utilisé pour afficher des textes en plus des vecteurs du champ.*

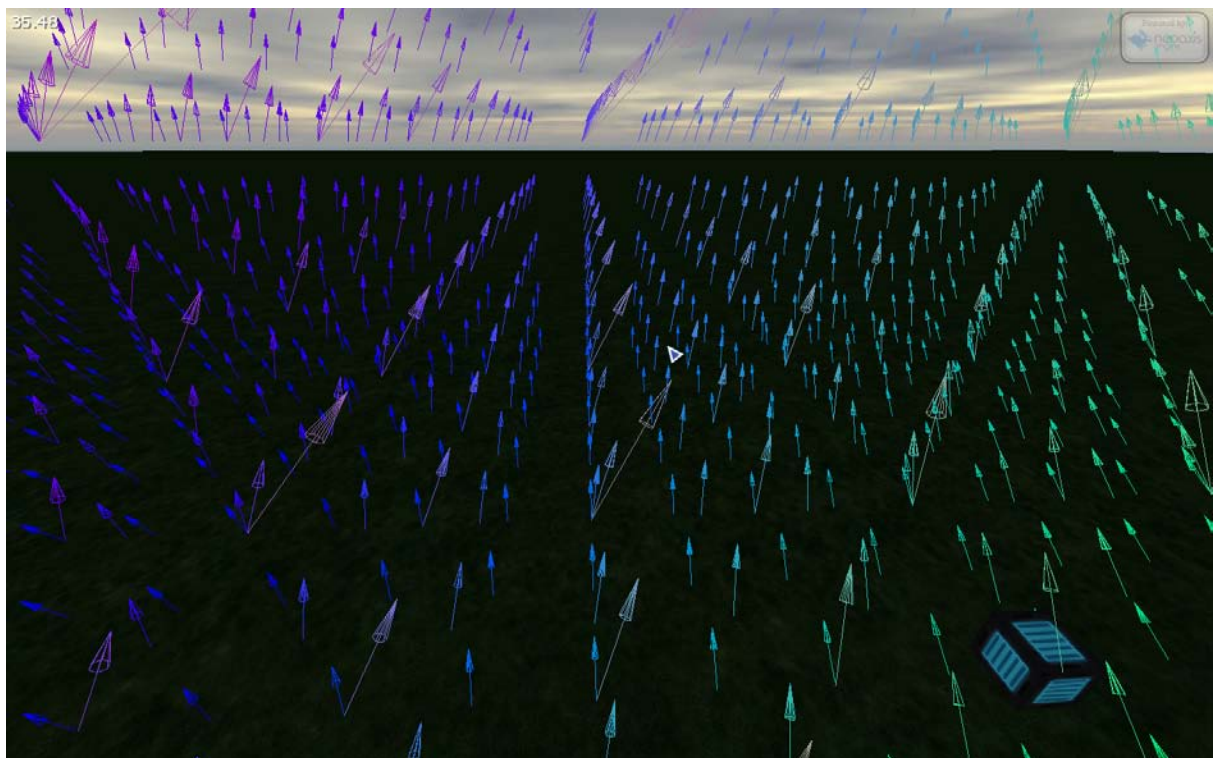
Dans cet exemple, la grille est fixe. Mais il est possible, en choisissant de place le centre du cube sur la position de la caméra, de déplacer le cube en même temps que la caméra. Pour cela, utilisez la ligne « `Vec3 o = c.Position` ». Le paramètre « Position » d'un objet donne sa position dans le repère XYZ de l'environnement 3D. De manière générale, tous les paramètres visibles dans l'éditeur de carte pour un objet (section « Properties » de l'éditeur de carte une fois l'objet sélectionné) peuvent être lus et écrits depuis un script C# associé à cette carte. Il est de même possible de modifier les propriétés d'un objet qui ont été définies via l'éditeur de ressources.

Dans le cas où le centre du cube est la position de la caméra, on a alors le rendu suivant :



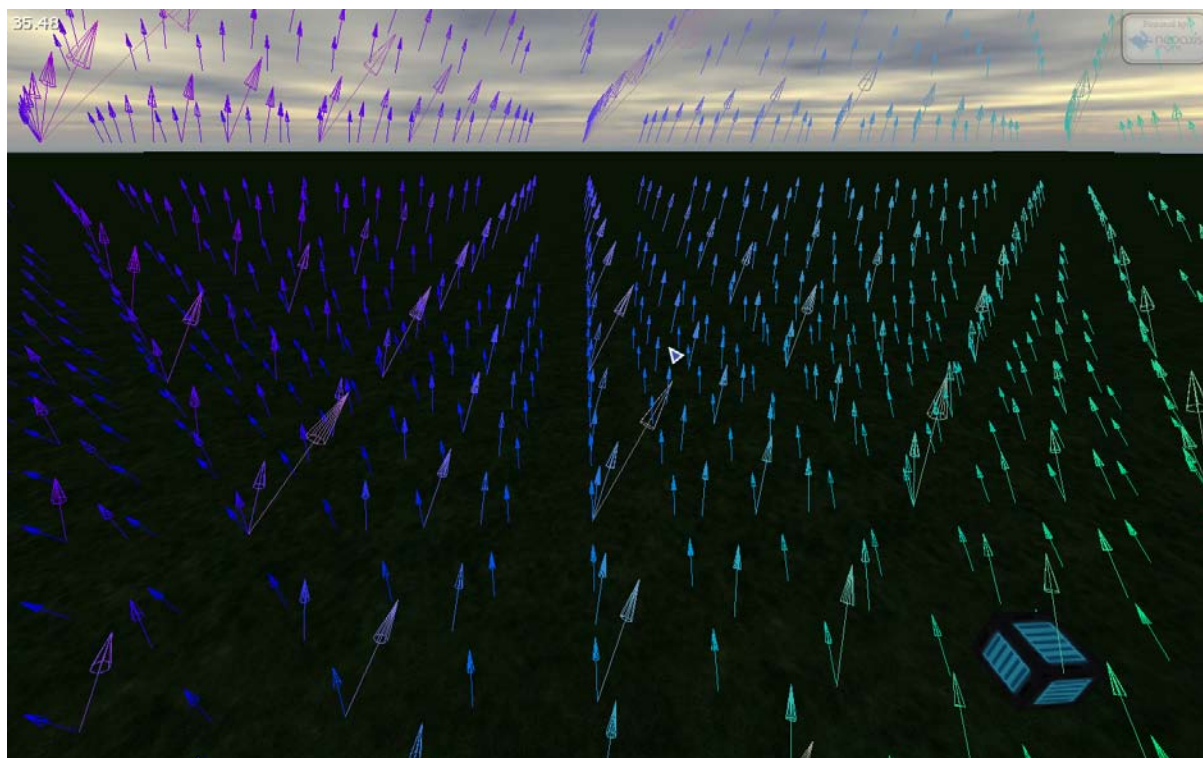


*Le centre du cube coïncide avec la position de la caméra...*



*... le cube contenant tous les vecteurs du champ « suit » donc la caméra.*

D'autres techniques peuvent être utilisées, par exemple, il peut être envisageable de créer une grille 3D pour tout l'environnement, et de ne calculer et afficher que les nœuds de la grille se trouvant à une certaine distance de la caméra, distance inférieure à une constante fixée :



*La grille peut être fixe et seuls les nœuds les plus proches de la caméra sont alors calculés et affichés.*

### Suite du travail

Deux méthodes d’affichage ont été présentées plus haut, mais ces méthodes ne sont pas réutilisables avec le code actuel. Il faudra donc concevoir un code C# plus souple, probablement au sein d’une classe statique dédiée, qui permettra de réutiliser les mêmes méthodes pour afficher les champs de vecteur.

Le choix de la méthode d’affichage est laissé à l’utilisateur. Donc, il sera à la charge du développeur de concevoir une classe permettant d’afficher un champ de vecteur suivant l’une (grille fixe, affichage des points les plus proches) ou l’autre (grille attachée à la position de la caméra) des deux méthodes. Il serait intéressant de faire en sorte que l’utilisateur puisse également choisir la formule qui permettra de calculer la valeur du champ de vecteurs en tout point de l’espace (en passant à la classe statique gérant les champs de vecteurs, un « pointeur sur la fonction », c’est-à-dire une méthode C# « delegate », calculant ce champ en tous points de l’espace par exemple).

Dans certains cas, on peut également ne pas vouloir calculer un champ de vecteur, mais simplement afficher des valeurs vectorielles (2D, 3D ou plus) dans l’environnement 3D. Par exemple, on peut connaître le flux de véhicules dans certaines rues, et on voudrait alors afficher une flèche de longueur et de couleur variables, dépendantes de ce flux. Mais il serait idiot de calculer le flux de voiture dans tout l’espace, puisqu’il se réduit aux routes. Il faudra donc que le développeur intègre, dans la classe gérant ces champs de vecteur, une ou plusieurs méthodes permettant d’afficher un ensemble fini et fixé de vecteurs, qui pourraient être issus d’une base de données (l’ensemble étant, par exemple, sous forme de liste chaînée ou de tableau).

Dans tous les cas, la méthode à utiliser sera « `Camera.DebugGeometry.AddArrow` », qui permet d’ajouter une « flèche » (un vecteur) à l’écran, dans l’espace 3D. La couleur se définit via « `Camera.DebugGeometry.Color` ».