7) Réalisation de « cartes de chaleur »

7.1) Introduction

Une carte de chaleur ou « *heatmap* » est une représentation graphique de données statistiques, qui fait correspondre à l'intensité d'une grandeur variable une gamme de nuances de couleurs.

Ce procédé permet de représenter visuellement, par exemple sur une carte géographique, des grandeurs numériques telles que des fréquences d'événements dans des zones données.

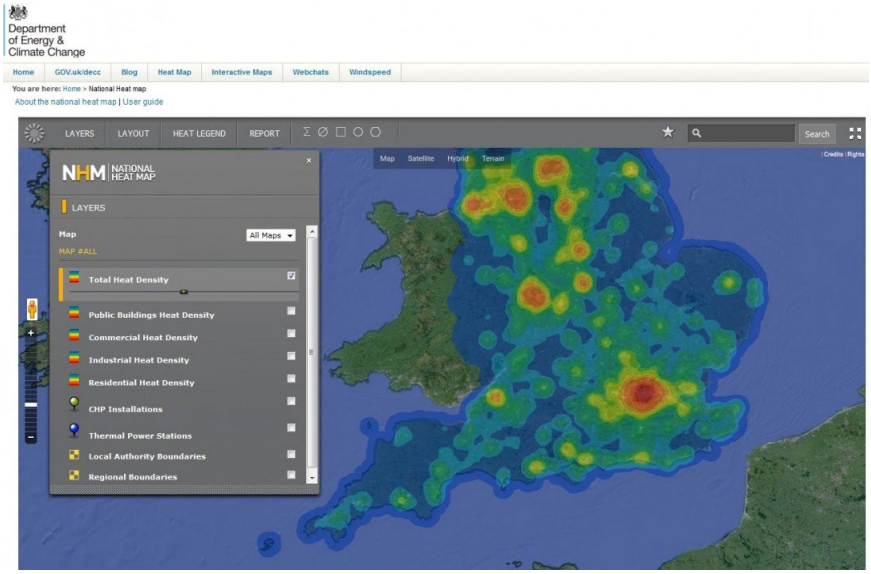

Figure 25 - densités de chaleur consommées par secteur en Angleterre

7.2) Localisation de points d'intérêts

Pour réaliser votre carte de chaleur, vous devrez calculer des grandeurs numériques pour chaque sommet de votre terrain texturé, qui seront leurs distances par rapport à certains aménagements.

Il est proposé à titre d'exemple d'utiliser deux distances, qui seront matérialisées par deux couleurs ; mais vous êtes encouragés à tester différentes combinaisons d'aménagements & couleurs de votre choix.

7.2.1) Requêter les données *Open Street Map*

Pour l'exemple, vous pouvez récupérer depuis le site web overpass-turbo-eu les données fournies issues des requêtes en langage *Overpass QL* suivantes :

```
[out:json][timeout:25][bbox:48.6935887,2.1504057,48.7201061,2.2187721];
(
  node["amenity"="bench"];
  node["leisure"="picnic_table"];
);
out body;
>;
```

```
out skel qt;
```

Cette première requête permet par exemple d'extraire des nœuds (*nodes*) de type « banc » ou « tables de pique-nique ».

```
[out:json][timeout:25][bbox:48.6935887,2.1504057,48.7201061,2.2187721];
(
  node["amenity"="bicycle_parking"];
);
out body;
>;
out skel qt;
```

Cette seconde requête permet par exemple d'extraire des nœuds (*nodes*) de type « parkings à vélos ».

Vous utiliserez le menu « Exporter » pour récupérer vos données au format *geoJson* que vous placerez dans votre sous dossier « *data* ».

7.2.2) Exploiter les données *Open Street Map*

Pour exploiter ces données, vous créerez une nouvelle classe *Poi* pour gérer les points d'intérêts.

Cette classe disposera d'une méthode *getPoints(String fileName)* qui retournera une liste de coordonnées d'aménagements pour un fichier donné.

7.3) Calculs des distances minimales

Pour chaque sommet de votre terrain créé dans la classe *Land*, vous devrez déterminer ses distances minimales à chaque type de point d'intérêt le plus proche, par exemple à l'aide de la fonction *dist()*.

Une fois calculées, vous devrez attribuer à chaque sommet les valeurs correspondantes afin de pouvoir les utiliser dans les *shaders*.

Vous pouvez attribuer des valeurs à un sommet lors de sa création avec la méthode *attrib(String name, float... values)* de la classe *PShape*. Le paramètre *name* est le nom de la variable qui sera reçue dans le *Vertex shader*, par exemple :

```
this.land.attrib("heat", nearestBykeParkingDistance, nearestPicNicTableDistance);
```

Vous pouvez également attribuer ces valeurs après la création de la *PShape* (*par exemple si vous affichiez des grandeurs dynamiques*) avec la méthode *setAttrib(String name, int index, float... values)* en précisant l'index du point à modifier.

Ces méthodes sont documentées dans la documentation technique de l'objet PShapeOpenGL.

7.4) Superposition de nuances à une texture

7.4.1) Traitement des sommets

Pour utiliser le mode d'édition des *shaders*, vous devez l'avoir installé au préalable à partir du menu *sketch / Importer une librairie / Ajouter une librairie* :

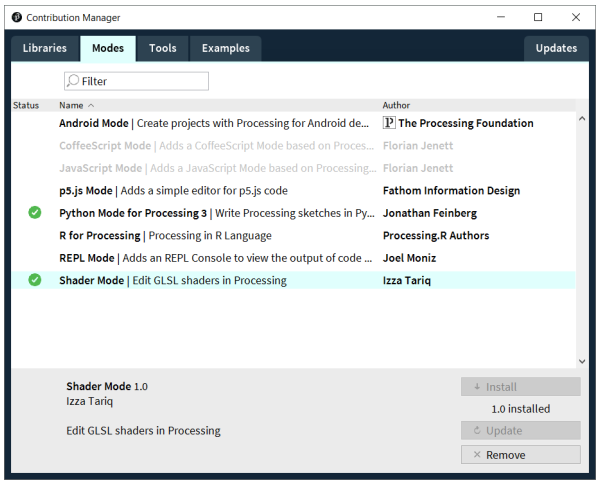

Figure 26 - Installation du mode Shader

Vous pouvez démarrer à partir d'un *shader* d'exemple. Comme il s'agit de mixer des nuances de couleur à la texture du terrain, vous pouvez démarrer à partir du *shader* obtenu par le menu *Shader / Shader Templates / texturevertex*.

Afin que vos attributs spécifiques soient transmis au *Vertex shader*, vous déclarerez un attribut « *heat* », ici de type *vec2*[17] puisqu'il contient deux valeurs flottantes, qui seront transmises au *Fragment shader* :

```
uniform mat4 transform;
uniform mat4 texMatrix;

attribute vec4 position;
attribute vec4 color;
attribute vec2 texCoord;
attribute vec2 heat;

smooth out vec4 vertColor;
smooth out vec4 vertTexCoord;
smooth out vec2 vertHeat;

void main() {
  gl_Position = transform * position;
  vertColor = color;
  vertTexCoord = texMatrix * vec4(texCoord, 1.0, 1.0);
  vertHeat = heat;
}
```

[17] La syntaxe « *varying* » étant dépréciée, on utilisera de préférence la syntaxe « *smooth out* » (vertex shader) et « *smooth in* » (fragment shader)

Vous pouvez ensuite intégrer votre *Fragment Shader* à partir de celui obtenu par le menu *Shader / Shader Templates / texturefragment* :

```
#ifdef GL_ES
precision mediump float;
precision mediump int;
#endif

uniform sampler2D texture;

smooth in vec4 vertColor;
smooth in vec4 vertTexCoord;
smooth in vec2 vertHeat;

void main() {
  gl_FragColor = texture2D(texture, vertTexCoord.st) * vertColor;
  // Insérez votre code ici
}
```

Vous devez ensuite modifier les composantes *RGB* des fragments *gl_FragColor.r*, *gl_FragColor.g* et/ou *gl_FragColor.b* en fonction des valeurs interpolées reçues dans *vertHeat[0]* et *vertHeat[1]* afin de modifier l'apparence de votre terrain.

Félicitations, vous savez désormais représenter des grandeurs à intensité variables sur un terrain représenté en trois dimensions :

Figure 27 - Carte de chaleur mixée à une texture

7.5) Optimisation des calculs (partie optionnelle)

L'algorithme de calcul des distances minimales a une complexité quadratique car il nécessite de comparer N sommets à M points d'intérêts. Une première optimisation évidente consiste à ne réaliser le calcul qu'une seule fois. Vous stockez les résultats par exemple dans un fichier au format JSON qui sera relu lors de la création du terrain.

Une méthode alternative consiste à générer à l'avance une image sur laquelle seront positionnés les points d'intérêts :

```
PGraphics ghm = createGraphics(5000, 3000, P2D);
ghm.smooth(8);
ghm.beginDraw();
ghm.hint(DISABLE_DEPTH_MASK);
ghm.hint(DISABLE_DEPTH_SORT);
ghm.hint(DISABLE_DEPTH_TEST);
ghm.hint(ENABLE_ASYNC_SAVEFRAME);
ghm.pushMatrix();
ghm.resetMatrix();
ghm.background(0xFF000000);
// Blend mode could be ADD, LIGHTEST or SCREEN depending on what we want to highlight
ghm.blendMode(SCREEN);
ghm.translate(2500.0f, 1500.0f);
// Insérez votre code ici pour dessiner les P.O.I
ghm.popMatrix();
ghm.save("data/heatmap.jpg");
ghm.endDraw();
```

Dans ce cas, vous dessinez les points d'intérêts avec un « halo » circulaire de proximité en utilisant des *PShape* en *TRIANGLE_FAN*, ce qui permet d'obtenir ce type d'image :

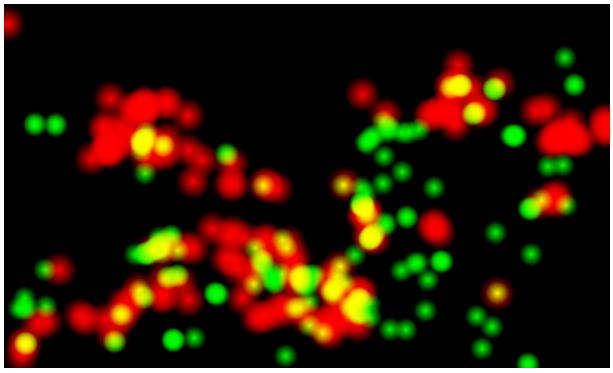

Figure 28 - Pré-calcul d'un calque de chaleur

Pour obtenir les distances souhaitées à fournir au *Shaders*, il suffit ensuite de charger cette image comme une texture avec la méthode *loadImage* de la classe *PImage.*

Ensuite vous chargerez les pixels individuels en mémoire avec la méthode *loadPixels*. La couleur de nuance d'un sommet positionné en *(x + 2500, 1500 - y)* [18] sera lue en récupérant la valeur de *pixels[x + y*largeur_image]*.

Vous pouvez passer cette couleur directement au *Vertex Shader* en utilisant un attribut *heat* de type *vec4* et utiliser ensuite les composantes souhaitées (ici *vertHeat.r* & *vertHeat.g*) dans le *Fragment Shader*.

Félicitations, vous avez mis en œuvre de nombreux concepts de représentation en 3D de zones géographiques, qui sont utiles à de nombreuses disciplines : géographie, géologie, urbanisme, épidémiologie, énergie, réseaux, militaire, etc. (*sans oublier les jeux vidéo*).

[18] Le pixel[0,0] se situe en haut à gauche d'une image, le pixel[image.width-1,image.height-1] en bas à droite.