IUT de Lannion
Dept Informatique

R5.A.06 - Prog multimédia
TP2 - Répartition des hôpitaux

P. Nerzic
2023-24

Créez un nouveau dossier pour ce TP : votre compte \ Multimedia \ TP2.

On repart des concepts des TD3 et TD4 concernant d3.js.

Le but du TP est de faire une carte montrant la densité des hôpitaux possédant un service des urgences en France. Les données sont tirées de OpenStreetMap.

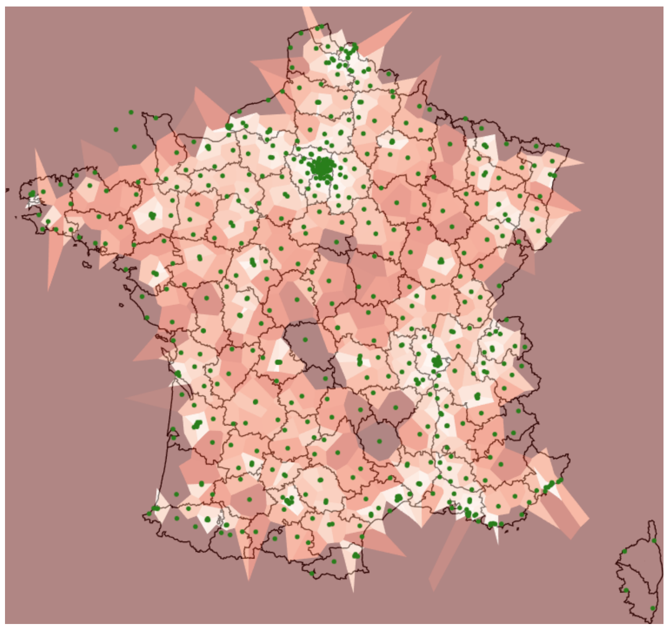

Figure 1: Résultat final

Concernant la notation, ce TP est en nombreuses étapes : étude de la norme GeoJSON qui représente les données, affichage de la carte de France, affichage des hôpitaux, gestion du zoom, ajout de bulles, affichage de la densité. Faites autant que possible, vous aurez des points pour chaque étape. Ne copiez pas les fichiers des collègues.

1. Conception générale

On part du fichier base_d3js.htm (clic droit, enregistrer la cible du lien sous...).

a. Renommez `base_d3js.htm` en `tp2.htm`. Ouvrez-le dans Visual Studio Code.
b. Modifiez la balise ainsi :

```
<svg xmlns=... id="dessin" width="800" height="800"></svg>
```

c. Programmez ceci dans la partie script :

```
const SIZE = 600

const svg = d3.select("#dessin")
```

d. Juste après ça, ajoutez une instruction qui ajoute à `svg` un attribut `viewBox` valant `"0 0 SIZE SIZE"` (`SIZE` étant remplacé dynamiquement par la constante).

2. Fichiers GeoJSON

C'est un format pour représenter des informations géographiques, normalisé par la RFC7946.

a. Allez voir l'exemple page 5 pour vous faire une idée du type de contenu.
b. Téléchargez departements.geojson. C'est un fichier décrivant les départements métropolitains.
c. Ouvrez-le avec Firefox. Vous allez voir que toutes les informations sont dans la propriété `features`. C'est un tableau de 95 objets `{type: "Feature", properties: {...}, geometry: {...}}`, un par département. Examinez-l'un d'eux. La `geometry` contient la liste des points du contour. Vous verrez que les coordonnées sont fournies sous la forme d'un tableau de deux réels, [*longitude*, *latitude*]. C'est important parce qu'il faudra les dessiner.
d. Téléchargez hopitaux.geojson qui servira aussi.
e. Constatez qu'il a la même structure, mais les *features* décrivent des hôpitaux. Celui de Lannion est le 914. On n'a pas beaucoup d'informations, car elles sont extraites de OpenStreetMap, voir data.gouv.fr. `emergency: "yes"` signifie qu'il y a un service d'urgence.

3. Affichage des départements

On va commencer par afficher le contour des départements avec d3.js. Ce sont les polygones des `features.geometry`. Heureusement, il y a une bibliothèque accompagnant d3.js qui facilite tous les dessins géographiques. Voici les concepts généraux.

- Les données à traiter sont des coordonnées [*longitude*, *latitude*]. Ces coordonnées doivent être projetées, c'est à dire transformées en (x, y) coordonnées sur l'écran (relatives à la *viewbox*).
- Dans d3.js, on définit une *projection* à l'aide du module d3-geo. Il y a une multitude de types de projections. L'une des plus connues et plus anciennes est celle de Mercator. Chacune possède ses qualités et défauts. Elles sont illustrées dans les sous-pages de d3-geo/projections.
- Une fois la projection définie, on dessine les points ou polygones des données.

Voici maintenant la réalisation.

3.1. Lecture des données

Il faut reprendre le principe des iris du TD3 pour lire le fichier `departements.geojson`.

a. Définir une fonction de dessin qui reçoit les données en paramètre.
b. Faire appeler cette fonction après la lecture du fichier (appel asynchrone)

3.2. Projection

a. Voici la projection qu'on va utiliser. Elle convient bien à la France :

```
const projection = d3.geoConicConformal()
    .center([2.454071, 46.279229])
    .scale(3000)
    .translate([SIZE/2,SIZE/2])
```

Vous pouvez la placer dans la fonction de dessin ou en dehors.

Analysez cette instruction complexe. Il y a l'appel à `d3.geoConicConformal()` qui retourne une fermeture. Ensuite, les appels à `center`, `scale` et `translate` qui sont des *setters* pour positionner la projection.

Le tout reste une fermeture, c'est à dire une fonction appelable. On lui fournit des coordonnées géographiques et elle retourne les coordonnées écran, par rapport à la *viewbox*.

b. Ajoutez temporairement ces deux lignes :

```
console.log(projection([2.454071, 46.279229]))
console.log(projection([-3.452665, 48.757893]))
```

La première ligne affiche la projection des coordonnées géographiques qu'on a fourni à `center`, donc, logiquement, les coordonnées écran sont celles du centre de la *viewbox*.

La deuxième ligne sont les coordonnées de l'IUT. Où est-ce sur l'écran ? Est-ce correct ?

La projection peut-être utilisée à l'envers. La fonction `projection.invert([x, y])` retourne la $[longitude, latitude]$ du point écran.

3.3. Dessin des contours

Il reste à dessiner les contours des départements. Ce sont des polygones dans le fichier GeoJSON. d3-geo permet de les transformer automatiquement en valeurs correctes pour l'attribut `d` de `<path>`.

a. Il faut générer des éléments `<path>` avec le mécanisme *selectAll.data.join* de d3, voir le TD3. Le type d'élément est `"path"`, les données sont `departements.features`.
b. Sur chaque `<path>` créé, il faut ajouter l'attribut `d` valant `d3.geoPath(projection)`.

Cette valeur d'attribut est une fermeture. Elle effectue le travail de création d'un *path* à partir des définitions de `departements.features`. Il faut comprendre que d3.js appelle automatiquement toute fonction ou fermeture qui se trouve en tant que valeur d'attribut, avec en premier paramètre, les données fournies à `data()`. Ici, ce sont les *features* des départements.

C'est à dire qu'en écrivant `.data(données).attr("a", fonction)`, d3.js appelle la fonction sur chaque donnée pour définir l'attribut. C'est un raccourci pour `.attr("a", d => fonction(d))`.

À ce stade, vous devriez voir la carte des départements s'afficher, mais tout en noir.

c. Ajoutez les attributs `fill` et `stroke` nécessaires pour un dessin des contours en noir et de l'intérieur en blanc.
d. Le code ci-dessus génère les chemins directement sous la balise `<svg>`. Faites en sorte qu'ils soient dans un `<g id="departements">` créé sous `<svg>`. En effet, il va y avoir plusieurs types de `<path>` qui ne doivent pas se mélanger.
e. Déplacez ce que vous avez fait en c. (les attributs constants `fill` et `stroke`) vers le groupe du d. En effet, la configuration de dessin étant identique pour tous les `<path>` doit se placer sur le groupe.
f. Le groupe ne doit pas être recréé s'il existe déjà. Voir le TD4 pour ajouter la fonction `appendIfAbsent` et l'utiliser.

4. Remaniement du logiciel

Il est temps de remanier le logiciel avant qu'il ne devienne trop gros. Il faut transformer les traitements en fermeture, comme dans le TD4. Voici les étapes :

a. Définir une fonction qui retournera la fermeture :

```
function DensiteHopitaux(size) {

  const projection = ... selon size

  function _dessinerDepartements(racine, departements) {
    // ajout des éléments <path> pour les departements.features
  }

  function dessiner(selection) {
    selection.each(
      function(datum) {
        _dessinerDepartements(d3.select(this), datum)
      }
    )
  }

  return dessiner
}
```

b. Affectez la variable globale `DH` avec `DensiteHopitaux(SIZE)`.
c. Faites appeler `DH` en lui fournissant les départements en tant que `datum`.

Rappel : `datum()` sert à fournir des données en bloc. Ce n'est pas comme `data()` qui gènère ou supprime des éléments pour coller aux données.

5. Dessin des hôpitaux

On complique un peu car on va traiter un second fichier de données.

a. Définir une autre sous-fonction privée, vide dans `DensiteHopitaux` :

```
function DensiteHopitaux(size) {

  const projection = ... en fonction de size

  function _dessinerDepartements(svg, departements) {
    ...
  }

  function _dessinerHopitaux(svg, hopitaux) {
  }

```

```
  function dessiner(selection) {
    selection.each(
      function(datum) {
        _dessinerDepartements(d3.select(this), datum)
        _dessinerHopitaux(d3.select(this), datum)   // OUPS !
      }
    )
  }

  return dessiner
}
```

On voit qu'il y a un problème de `datum` dans la fonction `dessiner()`. La solution, c'est que `datum` contienne les deux jeux de données, sous forme de propriétés distinctes. La variable `datum` sera un objet avec deux propriétés `departements` et `hopitaux`.

b. Corrigez en ceci :

```
      _dessinerDepartements(d3.select(this), datum.departements)
      _dessinerHopitaux(d3.select(this), datum.hopitaux)
```

c. Il faut maintenant charger les deux fichiers de données, en cascade :

```
function main() {
  // télécharger departements.geojson
  d3.json("departements.geojson")
    .then(departements => {
      // télécharger hopitaux.geojson
      d3.json("hopitaux.geojson")
        .then(hopitaux => {
          // dessiner departements et hopitaux
          d3.select("#dessin")
            .datum( {departements, hopitaux} )
            .call(DH)
        })
    })
}

main()
```

ou bien (à choisir) :

```
async function main() {
  // récupérer les données
  const departements = await d3.json("departements.geojson")
  const hopitaux     = await d3.json("hopitaux.geojson")

  // dessiner departements et hopitaux
```

```
  d3.select("#dessin")
    .datum( {departements, hopitaux} )
    .call(DH)
}

main()
```

Remarquez comment les deux groupes de données sont fournies ensemble à `datum()`. La syntaxe `{prop1, prop2, ...}` est un raccourci pour `{prop1: prop1, prop2: prop2, ...}` quand il y a des variables portant ces noms.

d. Il reste à programmer le dessin des hôpitaux, `_dessinerHopitaux()`. Comme pour les départements, il faut appliquer le mécanisme *selectAll.data.join*.
   - le type d'élément est `<circle>`
   - les données sont `hopitaux.features`
   - Il faut rajouter les attributs suivants :

```
    .attr("cx", f => projection(f.geometry.coordinates)[0])
    .attr("cy", f => projection(f.geometry.coordinates)[1])
    .attr("r", 2)
    .attr("fill", "green")
```

Le centre des cercles est le point projeté par `projection(f.geometry.coordinates)` ; cette fermeture retourne un tableau $[x, y]$. C'est dommage d'avoir à l'appeler deux fois, l'une pour récupérer x, l'autre pour y. On peut faire mieux. Il suffit de précalculer la projection des hôpitaux.

e. Transformez le code source précédent en ceci :

```
    .each(f => f.geometry.projected = projection(f.geometry.coordinates))
    .attr("cx", f => f.geometry.projected[0])
    .attr("cy", f => f.geometry.projected[1])
```

Elle définit une nouvelle propriété pour chaque *feature*, les coordonnées projetées, et elles sont utilisées sans recalcul pour définir les attributs `cx` et `cy`.

f. Comme pour les départements, il faut placer ces `<circle>` dans un groupe `#hopitaux` et y déplacer les attributs constants, sauf `r`.

6. Zoom sur le dessin

On va ajouter la possibilité de faire défiler et agrandir le graphique à la souris. C'est assez facile avec la bibliothèque d3-geo-zoom.

a. Inclure un script au début du HTML :

```
<!-- https://github.com/vasturiano/d3-geo-zoom -->
<script src="//unpkg.com/d3-geo-zoom"></script>
```

b. Complétez la fonction `dessiner()` :

```
function dessiner(selection) {
  selection.each(function(datum) {
    _dessinerDepartements(...)
    _dessinerHopitaux(...)

    // voir https://github.com/vasturiano/d3-geo-zoom
    d3.geoZoom()
      .projection(projection)
      .onMove(() => update(selection))
        (selection.node())
  })
}
```

Ces nouvelles instructions créent une fermeture assez complexe. Elle sert à modifier la projection et faire redessiner quand il y a une action avec la souris.

c. Avant la fonction `dessiner()`, ajoutez :

```
function update(selection) {
  selection.each(function(datum) {
    _updateDepartements(d3.select(this))
    _updateHopitaux(d3.select(this))
  })
}
```

d. Voici les deux nouvelles fonctions, à mettre avant `update()` :

```
function _updateDepartements(svg) {
  svg.select("#departements")
  .selectAll("path")
    .attr("d", d3.geoPath(projection))
}

function _updateHopitaux(svg) {
  svg.select("#hopitaux")
    .selectAll("circle")
      .each(f => f.geometry.projected = projection(f.geometry.coordinates))
      .attr("cx", f => f.geometry.projected[0])
      .attr("cy", f => f.geometry.projected[1])
}
```

En fait, elles sont presque comme leurs sœurs `_dessinerDepartements()` et `_dessinerHopitaux()`, mais ne font que modifier les attributs concernés par la modification de projection.

e. Il y a du code redondant entre `_dessinerHopitaux()` et `_updateHopitaux()` : la définition de certains attributs est identique. Alors faites en sorte, dans `_dessinerHopitaux()`, d'appeler `_updateHopitaux()` pour affecter ces attributs.
Idem entre `_dessinerDepartements()` et `_updateDepartements()`.

Vérifiez qu'on peut zoomer et déplacer la carte.

7. Bulles (*tooltips*) sur les hôpitaux

On veut faire afficher un rectangle d'information quand on survole un hôpital, pour afficher son nom. Il s'agit d'un `<div>` HTML qui est rendu visible et positionné à côté de la souris quand on passe sur un hôpital.

a. Définir un `<div id="tooltip"></div>` dans le document, après le `<svg>`.
b. Pour la bonne mise en page, ajoutez ce style :

```
<style>
  #tooltip {
    position: absolute;
    width: 400px;
    opacity:0;
    background-color: black;
    color: #fff;
    text-align: center;
    padding: 5px 0;
    border-radius: 6px;
}
</style>
```

c. Dans `_dessinerHopitaux()`, ajoutez des écouteurs sur les éléments `<circle>` :

```
      .on("mouseover", mouseover)
      .on("mousemove", mousemove)
      .on("mouseout", mouseout)
```

Voici le rôle de ces écouteurs :

- *mouseover* fait apparaître le `<div>`
- *mousemove* déplacer le `<div>` près de la souris
- *mouseout* cache le `<div>`

d. Ajoutez ces écouteurs sur les points représentant les hôpitaux :

```
function mouseover(event, f) {
  d3.select('#tooltip')
    .html(f.properties.name)
    .attr("hidden", null)
    .style("opacity", 1)
}

function mousemove(event, f) {
  d3.select('#tooltip')
    .style("left", `${event.pageX-200}px`)  // voir style #tooltip.width
    .style("top", `${event.pageY-40}px`)
}
```

e. Programmez `mouseout()`. Elle doit mettre l'opacité à 0 et `hidden` à vrai.

Il peut y avoir des défauts d'affichage des bulles, quand on bouge la souris trop vite.

8. Services des urgences

On voudrait ne voir que les hôpitaux ayant un service des urgences. Heureusement, c'est facile. Il suffit de filtrer les données lues du fichier des hôpitaux et ne garder que ceux qui ont la propriété `emergency: "yes"`.

C'est à faire dans la fonction `main()` quand on vient de récupérer les hôpitaux, et avant de les dessiner.

a. Lisez la documentation de la méthode `filter()` de `Array`. On doit fournir une lambda qui retourne `true` si on veut garder l'élément.
b. Le tableau à filtrer est `hopitaux.features`. Il faut traiter ce tableau et le remplacer par le résultat (réaffecter `hopitaux.features`).

9. Zones d'influence des hôpitaux

On veut afficher les zones d'influence de chaque hôpital. Ce sont des polygones qui entourent les points où un hôpital est le plus proche. Les contours de ces polygones sont les médiatrices des segments entre deux hôpitaux. On appelle cela un diagramme de Voronoi.

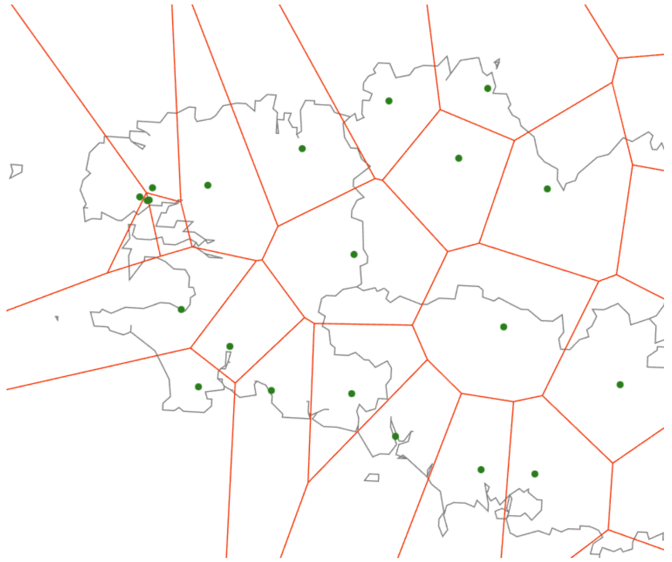

Figure 2: Cellules de Vonoroi

Plus la cellule d'un hôpital est grande, plus cet hôpital est isolé, plus il est loin des patients, si on considère une répartition homogène de la population. On veut donc remplir les cellules d'une teinte évoquant sa superficie.

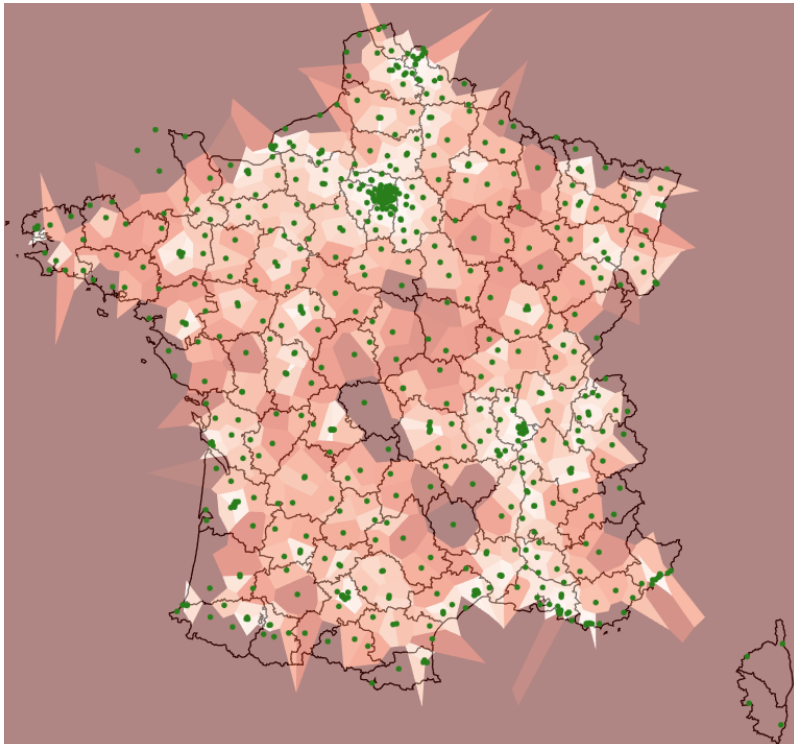

Figure 3: Densité des urgences

Il y a un souci avec les bords du dessin. Il faudrait limiter les cellules au contour extérieur, mais c'est très compliqué. Les cellules extérieures sont très étendues ; elles vont jusqu'à l'infini, par définition. Donc elles sont dessinées très sombres.

9.1. Dessin des cellules

Les cellules sont construites par une bibliothèque, d3-geo-voronoi spécialisée pour des calculs sur une sphère.

a. Inclure ce script au début du HTML :

```
<!-- https://github.com/Fil/d3-geo-voronoi -->
<script src="https://cdn.jsdelivr.net/npm/d3-geo-voronoi@2"></script>
```

b. Ajoutez ces deux nouvelles fonctions dans `DensiteHopitaux()` :

```
function _dessinerZonesHopitaux(svg, hopitaux) {
  // séparation en cellules de Voronoi au format GeoJSON
  const cellules = d3.geoVoronoi()
      .x(f => f.geometry.coordinates[0])
      .y(f => f.geometry.coordinates[1])
      (hopitaux)
      .polygons()

  // groupe pour les cellules
  svg.appendIfAbsent("#zoneshopitaux", "g")
```

```
      .attr("id", "zoneshopitaux")
    .selectAll("path")
      .data(cellules.features)
      .join("path")

  // mise à jour
  _updateZonesHopitaux(svg)
}

function _updateZonesHopitaux(svg) {
  svg.select("#zoneshopitaux")
    .selectAll("path")
      .attr("d", d3.geoPath(projection))
}
```

c. Configurez les modes de dessin : pas de remplissage, contours en rouge.
d. Faites appeler `_dessinerZonesHopitaux()` et `_updateZonesHopitaux()` comme leurs similaires. Attention, l'ordre d'appel dans `dessiner()` compte. La dernière fonction appelée dessinera ses élément au dessus des autres. Ça peut être intelligent de dessiner les cellules de Voronoi en dessous de tous les autres tracés, au moins au dessus des départements.

Vous devriez voir les cellules dessinées en rouge au dessus de la carte. Vous devriez toujours pouvoir zoomer et déplacer la carte.

9.2. Bulles sur les cellules

Les *tooltips* sont actuellement sur les hôpitaux. On voudrait les voir sur l'intégralité de la surface des cellules.

a. Déplacez la configuration des écouteurs vers `_dessinerZonesHopitaux()`. Ce sont les trois `on("mouse...", ...)`.
b. Dans `mouseover()`, remplacez le message affiché, `f.properties.name` par :

```
  .html(f.properties.site.properties.name)
```

C'est parce que la construction des cellules de Voronoi modifie la structure des données.

c. Pour voir les bulles, il faut configurer le mode de dessin *fill* `"white"` et *fill-opacity* à 0.1.

9.3. Coloration en fonction de l'aire

On arrive à la fin du TP. Il faut appliquer une couleur de remplissage des cellules. On va construire une sorte d'échelle qui relie une surface de cellule à une teinte. Comme pour les échelles, on définit un domaine, c'est à dire une plage de variation en entrée. Et on définit un *range* sous la forme d'un *interpolateur de couleur* associé à une gamme de couleurs. C'est expliqué sur cette page. Il y a de nombreuses gammes de couleurs, voir cet inventaire.

Voici les étapes. Tout se passe dans la fonction `_dessinerZonesHopitaux()`.

a. Calculez l'aire des cellules juste après les avoir définies :

```
// aire de chaque cellule
cellules.features.forEach(f => f.geometry.area = d3.geoPath(projection).area(f))
```

L'aire des cellules est comptée en nombre de pixels, car c'est avec les coordonnées projetées sur la *viewbox*. Si on redéfinit la *viewbox*, la projection changera et les aires aussi.

b. Définir une palette de couleurs :

```
const palette = d3.scaleSequential()
  .domain([0, 800])   // 800 = aire de la plus grande cellule (choix perso)
  .interpolator(d3.interpolateReds)
```

Le nombre 800 est une limite à l'aire choisie pour bien fonctionner, mais essayez d'autres valeurs. L'idée est que toutes les cellules dont l'aire est supérieure à cette limite sont de la couleur maximale dans la gamme.

`d3.interpolateReds` donne une gamme de rouges. Il y a d'autres gammes, voir la liste.

c. Définir la couleur de remplissage et la transparence de remplissage (à placer aux bons endroits) :

```
  .attr("fill-opacity", 0.5)
  .attr("fill", f => palette(f.geometry.area))
```