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

On repart des concepts d3.js vus dans le TD3.

1. Introduction

Le but global du TP est de faire un outil d'édition interactive de *partitions floues*. Une partition floue est composée de termes qui décrivent une information numérique de manière naturelle. Par exemple, si on s'intéresse à la température de l'eau de mer pour se baigner, on peut définir les termes flous « glaciale » , « froide » , « fraîche » , « tiède », « chaude » , « brûlante » . Tous ces termes constituent une partition, parce qu'ils décrivent toutes les températures possibles sans former de doublons. Il y a une définition plus rigoureuse, mais hors propos ici.

Ce sont des termes *flous*, c'est à dire qu'ils décrivent l'appartenance graduelle d'une valeur à ces termes. Par exemple, une température d'eau de 34°, ça vous paraît « chaud » pour vous baigner ? Oui tout à fait. Et 28°, est-ce toujours chaud ? Peut-être un peu moins. Et 24°, est-ce encore chaud ? Oui mais beaucoup moins, c'est davantage « tiède ». En plus, chaque personne a sa propre interprétation de ces termes.

Vous voyez que l'appartenance d'une température à un terme flou n'est pas booléenne, elle est graduelle. Chaque terme flou est défini par une fonction d'appartenance qui varie entre 0 (la valeur ne correspond pas du tout au terme flou) et 1 (la valeur correspond pleinement au terme flou).

Le contraire d'une appartenance floue est une appartenance booléenne : une température de 26° est soit totalement « chaude », soit ne l'est pas du tout. De plus, les frontières entre termes « tiède », « chaude » se font sur des valeurs fixes précises.

La manière la plus simple pour définir une fonction d'appartenance floue est d'utiliser un trapèze. Voici un exemple, figure 1, de trapèze qui définit une eau de mer « chaude ».

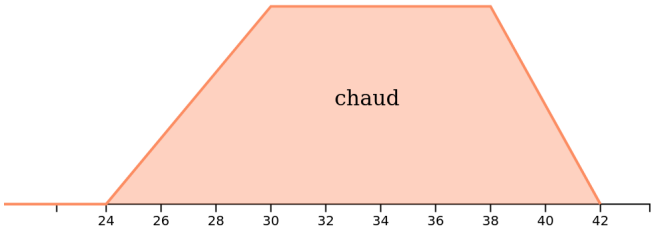

Figure 1: Terme flou « eau chaude »

Avec cette définition, en dessous de 24°, on ne parle pas du tout d'eau chaude, entre 24 et 30° la désignation d'eau chaude devient de plus en plus appropriée. Entre 30 et 38°, c'est de l'eau chaude. Au delà, ça commence à ne plus être de l'eau chaude, mais de l'eau brûlante (autre terme flou).

Le but est d'afficher une partition de termes flous, c'est à dire un ensemble de trapèzes, voir figure 2, puis de permettre des interactions simples à la souris, puis de la rendre éditable, de pouvoir déplacer les bornes en cliquant dessus.

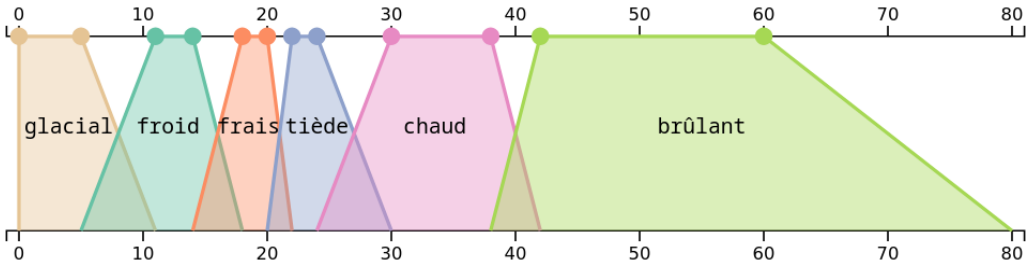

Figure 2: Résultat final

Concernant la notation, ce TP est en nombreuses étapes. Faites autant que possible, vous aurez des points pour chaque étape. Le barème sera adapté à ce qui aura pu être fait globalement. En aucun cas, pour aucune raison, ne copiez jamais les fichiers des collègues. Expliquez vos problèmes à l'enseignant, il vous excusera pour le temps que vous avez perdu.

2. Étape 1

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, avec le plugin *Live Server*.
b. Ajoutez ceci dans la partie script :

```
function PartitionFloue(trapezes) {
  console.log("TRAPEZES", trapezes)
}

const Trapezes = [
  { id: 0, supL:  5, corL: 11, corH: 14, supH: 18, nom: "froid" },
  { id: 1, supL: 14, corL: 18, corH: 20, supH: 22, nom: "frais" },
  { id: 2, supL: 20, corL: 22, corH: 24, supH: 30, nom: "tiède" },
  { id: 3, supL: 24, corL: 30, corH: 38, supH: 42, nom: "chaud" },
]
```

c. Ajoutez ce qu'il faut après la définition des trapèzes (tout à la fin), pour lancer la fonction `PartitionFloue` sur les trapèzes fournis. C'est plus simple que dans le TD3, car il n'y a pas de fichier à charger.

À ce stade, il doit seulement y avoir un message dans la console montrant les données à tracer.

3. Étape 2

Le but est de dessiner les trapèzes. Il faut mettre en place un système d'axe horizontal, et d'abord un système de coordonnées.

a. Dans la fonction `PartitionFloue`, affectez l'élément `<svg>` à une variable appelée `svg`.
b. Dans la fonction `PartitionFloue`, définissez une constante appelée `width` valant 600, et une constante `height` valant `width/4`.

On doit maintenant définir le système de coordonnées dans l'image svg, c'est à dire l'étendue des abscisses et des ordonnées : de 0 à `width` en x, et de 0 à `height` en y.

c. Dans la fonction `PartitionFloue`, programmez l'ajout de ce qu'il faut sur l'élément `<svg>` pour définir ce système de coordonnées.

Vous aurez moins de points si vous rajoutez cet attribut manuellement.

4. Étape 3

Le TD3 a montré pages 9 et 10 comment créer un système de coordonnées avec des marges ainsi que deux fonctions `fX` et `fY` pour traduire les valeurs des données en coordonnées dans l'image.

a. Ajoutez la constante `margin` valant `{top: 20, right: 10, bottom: 20, left: 10}`.
b. Définissez la fonction `fX` avec les caractéristiques suivantes :
   - type `scaleLinear`
   - domaine = `[ trapezes[0].supL - 1, trapezes[N-1].supH + 1 ]` avec `N` étant le nombre de trapèzes (constante à définir auparavant). Ce domaine permet de cadrer les trapèzes en laissant un peu de place avant et après.
   - range = de 0 à `width` en tenant compte des marges.
c. Définissez la fonction `fY` avec les caractéristiques suivantes :
   - type `scaleLinear`
   - domaine = `[0, 1]`,
   - range = de `height` à 0 en tenant compte des marges.
d. Il est très pratique de définir ces deux constantes :
   - `bottom` valant `fY(0)`. C'est l'ordonnée de la valeur 0.
   - `top` valant `fY(1)`. C'est l'ordonnée de la valeur 1.

À ce stade, il n'y a toujours rien de visible.

5. Étape 4

On va ajouter des graduations horizontales en bas et en haut pour faciliter la lecture des trapèzes.

a. Ajoutez des graduations horizontales en bas. Le code source est dans le TD3, page 10. Il est commode d'utiliser `bottom` au lieu de `height - margin.bottom` dans le `translate`.
b. Ajoutez des graduations horizontales en haut. C'est comme précédemment sauf que 1) le `translate` est sur `top`, et 2) la fonction s'appelle `d3.axisTop(fX)`.

NB: on n'ajoute pas d'axe vertical, parce que les trapèzes sont toujours entre 0 et 1.

On doit maintenant voir deux axes gradués horizontaux en haut et en bas. Les trapèzes seront dessinés entre eux.

6. Étape 5

Maintenant, on va dessiner les trapèzes.

a. Commencez par analyser les données. Les trapèzes sont définis par 4 coordonnées appelées `supL`, `corL`, `corH` et `supH`, voir la figure 3. Le support est la base du trapèze et le noyau

(*core* en anglais) est la partie où la fonction vaut 1. Pour chacun, il y a une borne basse (*low*) et une borne haute (*high*). Par exemple, `corL` est la borne basse du noyau et `supH` est la borne haute du support.

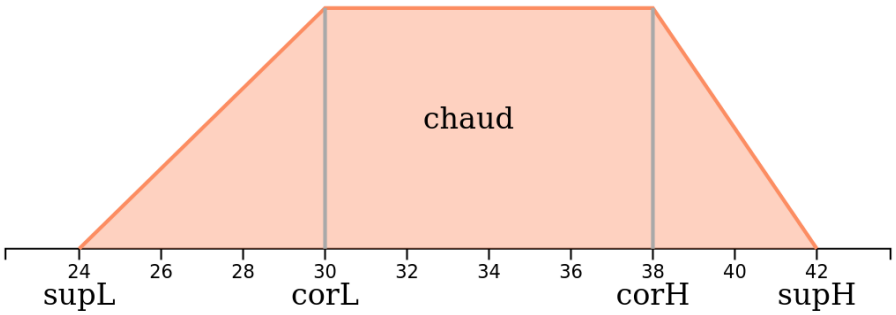

Figure 3: Support et noyau

Il faut donc dessiner une ligne qui passe par les points $(\text{supL}, 0)$, $(\text{corL}, 1)$, $(\text{corH}, 1)$, $(\text{supH}, 0)$, tous donnés en coordonnées *domaine* mais à traduire avec `fX` et `fY` en coordonnées dans l'image. Voici la démarche.

b. Faites générer un élément `path` pour chaque trapèze des données. Cet élément doit avoir un attribut `d` qui définit le chemin à dessiner. Relisez le TD1 concernant les chemins. C'est une succession de triplets `directive x y` avec `directive` valant `M` pour le premier point et `L` pour les suivants.

Si vous regardez le résultat, il est possible que plus rien ne marche correctement. C'est parce que le `<path>` que vous avez ajouté pour les trapèzes a vampirisé ceux des axes. Il faut donc créer un groupe `<g>` spécifique pour les trapèzes (vous y avez peut-être déjà pensé).

c. Définissez une constante contenant ce nouveau groupe : `grpTrapezes = svg.append("g")`
d. Remplacez votre `svg.selectAll("path")` par `grpTrapezes.selectAll("path")`.
e. Ajoutez plusieurs attributs sur chaque élément `<path>` :
   - couleur des contours : rouge
   - épaisseur des contours : 2
   - couleur de remplissage : gold
   - opacité de remplissage : 0.4

Consultez cette documentation sur les contours et cette documentation sur le remplissage

À ce stade, vous devez voir les trapèzes ainsi que les axes correctement dessinés.

7. Étape 6

Les couleurs des trapèzes ne sont pas agréables. On va y remédier. On voudrait que chaque trapèze ait sa propre couleur, tirée soit d'un tableau prédéfini, soit d'un dégradé proposé par d3.js.

La solution du tableau prédéfini semble la plus simple à première vue. Chaque trapèze possède un identifiant et on s'en sert comme index dans un tableau de `M` couleurs en utilisant l'index modulo `M`. Il faut ensuite appliquer cette couleur au contour et au remplissage du trapèze considéré, comme dans le TD3.

La solution d'un dégradé proposé par d3.js est plus intéressante mais plus compliquée conceptuellement. Pour commencer, il y a des palettes de couleurs intéressantes dans cette documentation. On s'intéresse particulièrement à `d3.schemeSet2` car les couleurs sont jolies, contrastées, ni trop claires, ni trop foncées. Elles iront à la fois aux contours et au remplissage des trapèzes. L'idée est de distribuer ces couleurs aux trapèzes.

L'attribution d'une couleur fonctionne comme les fonctions `fX` et `fY`. Il y a un domaine et un *range* (une étendue). Comme pour les fonctions `fX` et `fY`, on va construire une fonction `fColor` à laquelle on fournit une valeur du domaine, c'est à dire l'identifiant d'un trapèze, et elle retourne la valeur du *range* correspondant, c'est à dire la couleur.

a. Définissez une constante `fColor` qui est une fonction construite comme `fX`, mais telle que :
   - Le type d'échelle est `scaleOrdinal()`
   - Le domaine est celui des identifiants de trapèzes, des entiers. Il faut le spécifier soit par `.domain([0,1,2,3,4,..., N])` avec tous les entiers jusqu'à `N`, soit utiliser une fonction qui génère automatiquement cette liste d'entiers. Consultez la documentation de d3.range.
   - L'étendue est celle des couleurs, c'est à dire `d3.schemeSet2`.

b. Utilisez la fonction `fColor` pour définir la couleur des contours et remplissages de chaque trapèze.

À ce stade, les 4 trapèzes ont chacun une jolie couleur.

8. Étape 7

On voudrait ajouter les noms des termes flous au centre de chaque trapèze. Il n'est pas possible de rajouter plusieurs éléments simultanément dans le cadre d'un `selectAll-data-join` simple. Voici comment faire. Il y a plusieurs points.

- D'abord, quand on doit créer plusieurs éléments par donnée, ici un `<path>` et un `<text>`, on doit impérativement les placer dans un groupe. Donc le schéma de base de d3.js doit créer des éléments `<g>`, un par trapèze.
- Ensuite, le principe est d'utiliser la variante de `join` où il y a 3 lambda, mais ici, on n'a besoin que de la première, celle qui s'appelle *enter*. Dans le corps de cette lambda, on peut créer autant de sous-éléments qu'on veut.

a. Voici le schéma général, adaptez-le et complétez-le : 

```
grpTrapezes
  .selectAll("g")
  .data(...)
  .join(
    enter => {
      const g = enter.append("g")
      g.append("path")
        .attr(...)
        ...
      g.append("text")
        .text(trp => trp.nom)
```

```
    }
  )
```

Explication: `enter` désigne l'emplacement où on fait les ajouts, c'est à dire l'élément `<g>` pour chaque trapèze.

b. Il manque au moins deux attributs, `x` et `y` à l'élément `text` pour s'afficher. Définissez-les comme vous voulez, mais il serait judicieux de centrer le texte par rapport à son trapèze. Pour un meilleur centrage, définissez l'attribut `text-anchor` valant `middle`. Vous aurez avantage à déplacer cet attribut vers le groupe `grpTrapezes`.
c. Vérifiez la structure SVG avec l'inspecteur du navigateur. Pour chaque trapèze, il doit y avoir un `<g>` contenant un seul `<path>` et un seul `<text>`. Ça ne doit pas être le Zoo de Thoiry.

À ce stade, on a fini l'affichage. Les trapèzes sont bien visibles et avec leur nom.

9. Étape 8

Tout ne va pas si bien que ça. Si on relance la fonction de dessin, `PartitionFloue` avec de nouveaux trapèzes, par exemple, ça va très mal se passer.

a. Ajoutez ceci tout à la fin et constatez les dégâts :

```
const Trapezes2 = [
  { id: -1, supL: 0, corL:  0, corH:  5, supH: 11, nom: "glacial" },
  ...Trapezes,    // ajout ici des éléments de Trapezes
  { id: 4, supL: 38, corL: 42, corH: 60, supH: 80, nom: "brûlant" },
]

PartitionFloue(Trapezes2)
```

Tout est en double. Les axes et les trapèzes. Pourquoi ? Parce qu'on a systématiquement créé de nouveaux éléments, par exemple les graduations, sans vérifier s'ils étaient déjà présents.

Il va falloir remplacer tous les `svg.append("EE")` par `svg.appendIfAbsent("#idEE", "EE").attr("id", "idEE")`. La fonction `appendIfAbsent` est programmée au début du fichier. C'est quelque chose qui manque dans `d3.js`, de n'ajouter un sous-élément `<EE>` dans un autre que s'il est absent. Elle se base sur le sélecteur CSS fourni, ici c'est un identifiant. Donc il faut ensuite affecter cet identifiant, afin que ce sous-élément ne soit pas recréé par la suite.

a. Faire cette transformation sur les graduations, ainsi que sur `grpTrapezes`, en choisissant des identifiants appropriés.

Ça ne résout que le problème des axes ; on ne voit plus les doubles graduations. Par contre, les trapèzes sont toujours en double. Ce second problème vient du fait qu'on n'a pas mis à jour les trapèzes déjà dessinés. Or la fonction `fX` a changé car les données vont de 0 à 80 alors qu'avant c'était de 5 à 42 seulement. Il faut programmer la lambda *update*. Elle doit faire comme la lambda *enter* à part créer les éléments. Relire les exercices 1.3 et 1.4 du TD3.

b. Ajoutez la lambda *update* dans le dessin des trapèzes. Elle doit sélectionner les `<path>` et les `<text>` et réaffecter tous les attributs et les textes, sauf ceux qui donnent une valeur constante. Voici une esquisse de ce qu'il faut faire :

```
grpTrapezes
  .selectAll("g")
  .data(...)
  .join(
    enter => {
      const g = enter.append("g")
      g.append("path")
        ...
      g.append("text")
        ...
    },
    update => {
      update.select("path")
        .attr(...)
      update.select("text")
        .attr(...)
    }
  )
```

Dans la lambda *update*, le paramètre `update` contient l'élément `<g>` associé à un trapèze.

c. Pour finir, avez-vous pensé à associer un identifiant aux données ? Ça a été signalé dans le TD3, exercice 1.4.
d. Vérifiez que tout fonctionne en ré-appelant une dernière fois `PartitionFloue(Trapezes)` tout à la fin du script. Les trapèzes doivent revenir comme au début. Les curieux remarqueront que `PartitionFloue([])` plante, mais c'est un cas qui serait difficile à traiter et donc qui n'est pas demandé.

Maintenant, les trapèzes sont correctement affichés après modification des données.

10. Étape 9

On voudrait pouvoir zoomer et faire défiler le graphique à la souris. Il y a un mécanisme tout prêt dans d3.js, documenté sur cette page mais c'est un peu compliqué au premier abord. En fait, c'est beaucoup plus simple quand on utilise des fermetures, mais on ne le fait pas dans ce TP. Alors voici le principe qu'on va employer.

1. On rajoute un écouteur géré par d3.js sur les clics+glissés souris et les actions avec la roulette.
2. Cet écouteur est activé à chaque mouvement utile de la souris et il fournit une « transformation ». Il s'agit d'une matrice, mais on n'a pas besoin de le savoir. Elle représente le changement de vue sur le graphique.
3. On doit modifier `fX` par cette transformation, puis rappeler la fonction `PartitionFloue`.
4. La fonction `fX` modifiée change le calcul des coordonnées, ce qui produit l'effet de zoom et de défilement.

Voici le détail de ce qu'il faut programmer.

a. Ajoutez un second paramètre `transform=null` à la fonction `PartitionFloue`.
b. Plus bas, on a `const fX = d3.scaleLinear()`, changez `const` en `let`.
c. En dessous de ces trois lignes concernant `fX`, programmez cette conditionnelle :

   `si transform n'est pas null, alors fX = transform.rescaleX(fX)`

   C'est ça qui applique la translation et homothétie de `transform` sur l'axe des x.
d. Il faut lui ajouter une clause *sinon* contenant ces instructions qui installent l'écouteur : 

```
const zoom = d3.zoom()
  .extent([[0, 0], [width, height]])
  .scaleExtent([0.25, 64])
  .on("zoom", ({transform}) => PartitionFloue(trapezes, transform))
svg.call(zoom)
```

On se rend compte que toute la fonction `PartitionFloue` est rappelée à chaque mouvement souris, et on refait tous les axes et autres. C'est du gâchis. Avec une fermeture, on ne redessinerait que le minimum.

11. Étape 10

On continue à améliorer le logiciel en ajoutant la possibilité d'éditer les trapèzes. On doit pouvoir déplacer les bornes des noyaux (la partie du haut des trapèzes). On va ajouter des points de contrôle comme dans la figure 4.

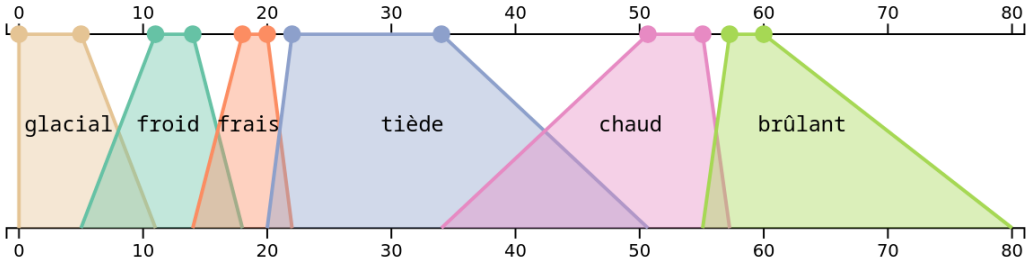

Figure 4: Édition des noyaux

a. Là où vous créez les `<path>` et `<text>` de chaque trapèze, ajoutez la création d'un cercle de rayon 5 et positionné au niveau de `corL`. Sa couleur est celle du trapèze, contour et remplissage. Faites de même pour `corH`.

Vous devez aussi gérer le cas de la mise à jour, sinon quand vous zoomerez à la souris, les points ne suivront pas. Mais il y a un souci parce que les deux cercles sont indiscernables. Il faut leur ajouter une propriété qui les identifie. On ne peut pas utiliser un identifiant car il y a ces deux points sur chaque trapèze. On peut utiliser une classe.

b. Quand vous créez les cercles, ajoutez-leur une classe comme `corL` et `corH`. Vous pouvez utiliser `.attr` ou `.classed("nom", true)`.
c. Il faut programmer le cas de la mise à jour des trapèzes. Il suffit de réaffecter la position `cx` des cercles.

Enfin, on va ajouter un écouteur pour les clic+glissé sur ces cercles. Il y a d'une part un écouteur à installer et une fonction à programmer. L'écouteur est ce que d3.js appelle un comportement (*behaviour*), et c'est documenté dans d3-drag. C'est compliqué à comprendre mais simple à utiliser.

d. Appelez l'instruction suivante à la fois quand vous créez un cercle et quand vous le mettez à jour. Mettez `onDragCorH` pour l'autre cercle.

```
    g.append("circle")
        ...
        .call(d3.drag().on("drag", onDragCorL))
```

`onDragCorL` et `onDragCorH` sont deux fonctions qui modifient les trapèzes. Voici comment les programmer à l'intérieur de `PartitionFloue` :

```
function onDragCorL(event, trp) {
  // nouvelle coordonnée du point
  let x = fX.invert(event.x)
  // test de validité
  if (x < trp.supL || x > trp.corH) return
  // prise en compte de la coordonnée
  trp.corL = x
  //optionnel: if (trp.prec !== null) trp.prec.supH = x
  // tout redessiner
  PartitionFloue(trapezes, transform)
}
```

e. L'autre fonction est similaire. Mettez-les en place toutes les deux.

Ces deux fonctions vérifient qu'on ne déplace pas le point au delà du support et de l'autre limite du noyau. En fait, il y a maintenant quelque chose de très compliqué à faire, mais qui est optionnel. Normalement, dans une partition floue forte, les trapèzes successifs sont fortement liés. Le point `supH` d'un trapèze est lié au point `corL` du trapèze de droite, et le point `supL` d'un trapèze = le `corH` du trapèze de gauche.

Pour permettre cela, il faut que chaque trapèze de la liste contienne une référence sur son prédécesseur, `null` si aucun, et son successeur, `null` si aucun. Il y a donc un prétraitement à faire sur la liste. Saurez-vous le faire ? C'est un premier parcours dans lequel on met ces liens `prec` et `next` à `null`. Puis un second parcours dans lequel on se souvient du précédent trapèze vu (null au début) et on met à jour les liens à la fois du courant vers le précédent, et du précédent vers le suivant.

Ensuite il reste à décommenter l'instruction qui modifie le trapèze voisin.

12. Remise du travail

Déposez votre fichier `tp2.htm` sur Moodle dans la zone de dépôt du TP. Vous ne devez en aucun cas copier le travail, même partiel d'un camarade. Les enseignants sont là pour vous aider à avancer du mieux possible, mais sans vous donner directement la solution. Si vous avez rencontré un problème, ajoutez une balise `<p>explications...</p>` dans votre fichier `tp2.htm` où vous expliquez rapidement ce qui s'est ou pas passé.