Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

L2 Info - 2022-2023
Algo des Arbres
Devoir n°2

Segmentation par QuadTree

L'objectif est de vous faire manipuler une structure réductrice et/ou accélératrice classique dans de très nombreux problèmes de partitionnement et/ou segmentation bi-dimensionnelle : le QuadTree.
Pour ce DM, le projet du 1er semestre est un bon point de départ : beaucoup d'éléments sont les mêmes.

Préambule

Le principe est simple : on part d'une zone de données bidimensionnelles (une portion de surface, une image...), de tailles connues (appelons les `W` et `H`) et on la découpe en 4 sous-zones, de tailles égales(`W`/2,`H`/2). Puis on recommence récursivement sur chacune des 4 sous-zones.

Les critères de subdivision/arrêt peuvent être variés mais on peut les résumer à trois situations simples :

ⓐ on a atteint une taille limite, portion d'espace qu'on ne peut plus découper (par exemple un `pixel`).

ⓑ la zone est *vide* ou *uniforme* : le noeud père et ses 4 noeuds fils sont identiques, il n'y a plus d'information à "segmenter".

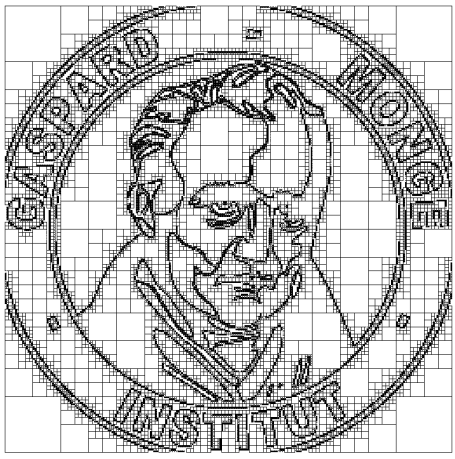
ⓒ une condition prédéfinie est vérifiée (présence/identification d'un "objet" particulier...)

☞ Souvent, comme dans ce DM, les 3 situations coexistent...

Quelques exemples classiques

- Image numérique

  Une image est une zone bidimensionnelle formées de `pixels` (carrés insécables). Lorsque que 4 pixels voisins sont identiques on peut les grouper et condidérer simplement que l'on a affaire à un `pixel` 'plus gros'.
  Ainsi une grande zone uniforme pourra être traitée comme un seul très gros `pixel`.
  Les applications sont nombreuses et variées : compression, recherche de formes (*pattern*), extraction de contours...
  Dans ce cas, la hauteur maximale de l'arbre est simple : on s'arrête lorsque la taille du noeud est réduite à 1 `pixel` ou lorsque la zone est considérée comme *uniforme*.

  ☞ dans ce genre d'application, le QuadTree est utilisé comme une structure *réductrice*.

- Réduction de complexité (le cas qui nous intéresse ici)

  Une autre usage classique des QuadTree est de réduire la complexité de certains types de calculs.

  Considérons par exemple un ensemble de `N` *particules* en interaction (disons de simples points en mouvements, ou une armée de trolls de l'espace dans un film ou un jeu video). Pour gérer toutes les interactions possibles il faut, au minimum, calculer les distances entre chaque paire de *particules*. La complexité est alors à l'évidence en $o(N^2)$.

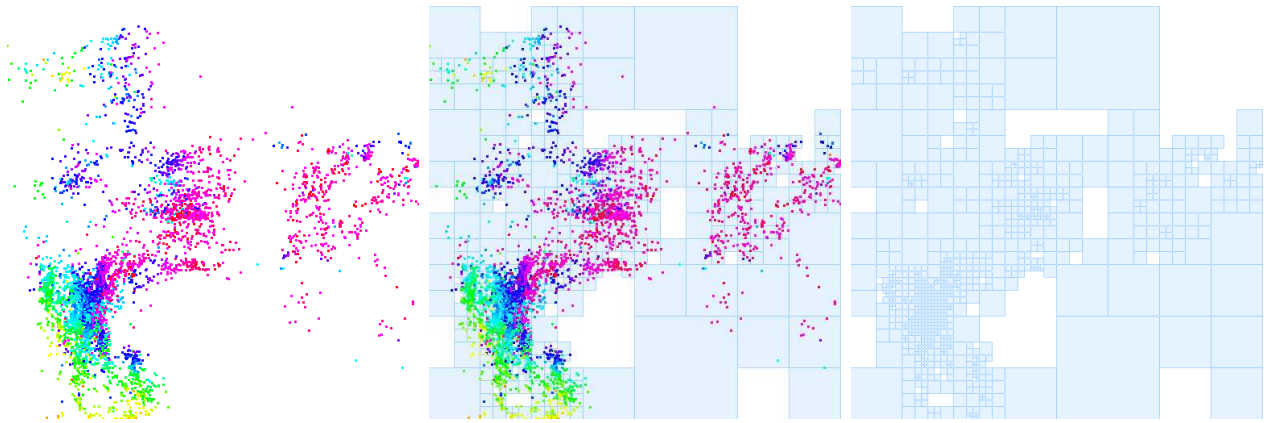
  Le QuadTree permettra de réduire drastiquement cette complexité en considérant, par exemple, que les interactions n'ont lieu qu'entre *particules* suffisament proches et/ou qu'un *troll* ne peut interagir qu'avec un nombre limité de voisins.

On découpe l'espace en zones (le QuadTree) de tailles limitées (distance d'interaction) et ne contenant qu'un nombre réduit d'individus (les voisins possibles), et on ne calcule les interactions qu'à l'intérieur d'une zone et entre zones connexes (qui se touchent).

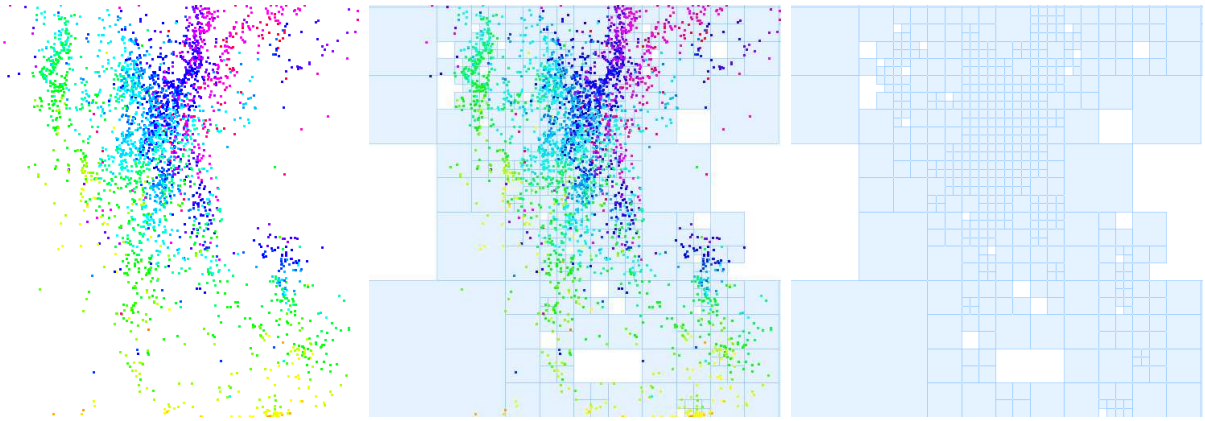
L'espace entier est alors formé de p zones contenant au plus k (très inférieur à N) individus. La complexité est alors réduite, en gros, à $o(p * k^2)$.

☞ dans ce cas, le QuadTree est utilisé comme une structure *accélératrice*.

Dans l'exemple ci-dessous, avec N = 2000 points en interaction dans une zone de taille (512×512), le QuadTree segmente l'espace en zones de taille min 8×8 contenant au plus 10 points (les zones blanches n'en contiennent aucun).



☞ zoom sur le quart inférieur gauche, particulièrement dense :



Une *feuille* du QuadTree doit donc satisfaire deux conditions (pas toujours compatibles...) :

① la zone est au moins de taille 8x8

② si la condition ① est vérifiée, la zone contient au plus 10 points

☞ C'est sur ce second type d'application que porte ce devoir. Il ne s'agit pas de gérer les interactions, mais simplement de mettre en place la subdivision par QuadTree sur un nuage de points, afin d'*isoler* les points seuls ou en petits groupes, dans les noeuds du QuadTree. Le reste, c'est pour les années suivantes...

☞ Par de nombreux aspects, en particulier la partie graphique (à réaliser avec la libMLV) ce projet présente des similitudes avec ce que vous avez fait au semestre précédent pour les enveloppes convexes. Vous pouvez bien évidemment vous en resservir comme base de départ (programmer c'est savoir capitaliser et recycler !).

Les données du problème

Dans ce genre d'application, gourmande en calcul, il est important de bien *calibrer* le problème en amont pour éviter les opérations non indispensables.

Gestion de la mémoire

☞ En particulier nous limiterons ici les appels aux fonctions de gestion de mémoire (`malloc/free`) au strict minimum.

Les données de départ sont :

- ⓐ la taille initiale de la zone de travail : pour des raisons évidentes nous partirons d'une zone *carrée* dont la dimension (entière) est une puissance de 4 : `WxH` = 512x512 ou 1024x1024 (ça correspond à des `pixels`).
- ⓑ la taille minimale (taille d'une *feuille* (w_{min}xw_{min}), arrêt de subdivision) sera elle aussi une puissance de 4 ((1x1), (2x2), (4x4)...).
- ⓒ le nombre N_p de *particules* peut quant à lui être quelconque, de même que le nombre max. (K_p) de particules contenu dans une *feuille*. Ces particules seront stockées dans un tableau.

Des conditions ⓐ et ⓑ découle le fait que la hauteur totale du QuadTree est parfaitement **connue** dès le départ : c'est $h = \log_2(W)$.

☞ Le QuadTree sera **toujours** initialisé sous sa forme complète, c'est à dire comme si il allait être utilisé jusqu'à sa plus fine résolution ($w_{min} = 1$). On peut donc évaluer la taille mémoire nécessaire et stocker le QuadTree sous la forme d'un tableau de `noeuds` structuré comme un "tas d'ordre 4".

☞ Les paramètres principaux sont donc $W = 2^h$ et N_p et il n'y a que 2 allocations de mémoire à produire : le QuadTree et le tableau de particules (aucun `noeuds` ni aucune particule ne sera créé(e) ou détruit(e) en cours de route). Et en paramètres secondaires on fixera la taille minimale d'une feuille w_{min} et le nombre maximal de particules qu'un `noeud` peut accueillir K_p.

Structures de données

- ⓐ **les particules** : au départ ce sont de simples points rangés dans un tableau de taille prédéterminée (paramètre du programme). Ces points seront initialisés soit en bloc, de manière aléatoire, soit un par un au clic souris.

  ☞ Une particule ne connait rien du QuadTree ni de ses voisines. Tout ce qu'elle connait c'est sa position (éventuellement sa vitesse si on veut la faire bouger).

  ☞ Elles constituent les **données primaires** du problème, sur lesquelles vient se greffer la structure accélératrice du QuadTree.

- ⓑ **chaînage des particules** : en parallèle du tableau de points, il sera nécessaire de maintenir un système de *chaînage* qui permettra d'accéder facilement à la liste des particules contenues dans une feuille du QuadTree. Les points pouvant théoriquement se déplacer, il peuvent changer de `noeud` du QuadTree. Et même si les points ne bougent pas, à l'initalisation du système ils sont intégrés un par un dans le QuadTree.

  Lorsque qu'un `noeud X` (non terminal) contient K_p points (la limite fixée en paramètre) il est saturé.

  ☞ Si on lui en rajoute un il "déborde" et sa liste doit être **purgée** (vidée) : les $(K_p + 1)$ points sont alors **dispatchés** sur ses fils, et ainsi récursivement jusqu'à ce qu'aucun noeud ne contienne plus de K_p points. Le `noeud X` perd son statut de `feuille` du QuadTree au profil d'au moins 2 de ses descendants (ça peut être plus – cf. exemple en p4).

  ☞ Si les $(K_p + 1)$ points sont très proches, la *purge récursive* peut les faire descendre de plusieurs étages dans le QuadTree, jusqu'à ce qu'un point au moins se sépare des autres. On aura alors K_p points dans une `noeud` et 1 dans un `noeud` frère.

  Seuls les `noeuds` terminaux (de taille w_{min}) ne sont pas concernés par cette procédure et pourront contenir un nombre illimité de points.

On utilisera donc une structure de liste contenant un pointeur vers un point (toujours le même), et un lien vers une cellule pointant sur un autre point du même `noeud`. Ces cellules seront stockées dans un tableau, parallèle au tableau des points.

ⓑ **le** QuadTree : un `noeud` du QuadTree sera une structure contenant au minimum

- des pointeurs vers ses 4 `noeuds` fils (ordre arbitraire),
- des informations géométriques pour le situer dans l'espace (à vous de choisir),
- une liste `plist` de cellules pointant vers les `particules` présentes dans le `noeud`.
- un entier `nbp` indiquant le nombre de particules *«couvertes»* par le noeud.

Attention : le champs `nbp` **ne correspond pas** à la longueur de la liste `plist` mais au nombre de `particules` situées *«géométriquement»* **sous** le noeud, c'est à dire soit dans le noeud lui-même, soit réparties dans chacun de ses fils.

- un noeud `a` tel que $a \rightarrow nbp = 0$ est un noeud vide (et forcément $a \rightarrow plist = NULL$),
- un noeud `b` tel que $b \rightarrow nbp \neq 0$ est
  - soit un noeud interne (**alors** $b \rightarrow nbp > K_p$ **et** $b \rightarrow plist = NULL$),
  - soit une feuille (**alors** $b \rightarrow nbp \leq K_p$ **et** $a \rightarrow plist \neq NULL$).

  ☞ le champs `nbp` de la racine est le nombre total de particules (cf. exemple suivant).

☞ Les listes propres à chaque `noeud` constituent l'*interface* entre les données primaires (les points) et la structure accélératrice (le QuadTree).

Chaque particule est connue du `noeud` dans lequel elle se trouve, et se trouve ainsi liée (via la liste) aux autres particules présentes dans `noeud`. Ce sont ces mini-listes qui définissent l'état de *voisinnage* des particules et permettrait la mise en oeuvre des calculs d'interaction :

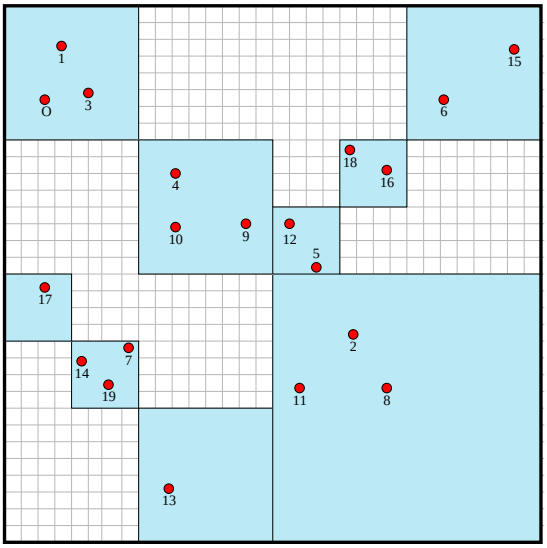
- en parcourant l'arbre on trouve un `noeud` (une feuille) contenant $k_1 (< K_p)$ particules. Ces particules sont *voisinnes* donc on effectue les calculs de complexité quadratique (mais sur k_1 particules seulement).
- puis on trouve un `noeud` **adjacent** contenant d'autres particules (autre liste de k_2 particules). On a ainsi deux petits groupes de particules sur lesquels on effectue un calcul de complexité $O(k_1 * k_2)$.
- et on fait ça pour tous les couples de `noeuds` adjacents du QuadTree, indépendemment de leurs tailles (niveau dans l'arbre).

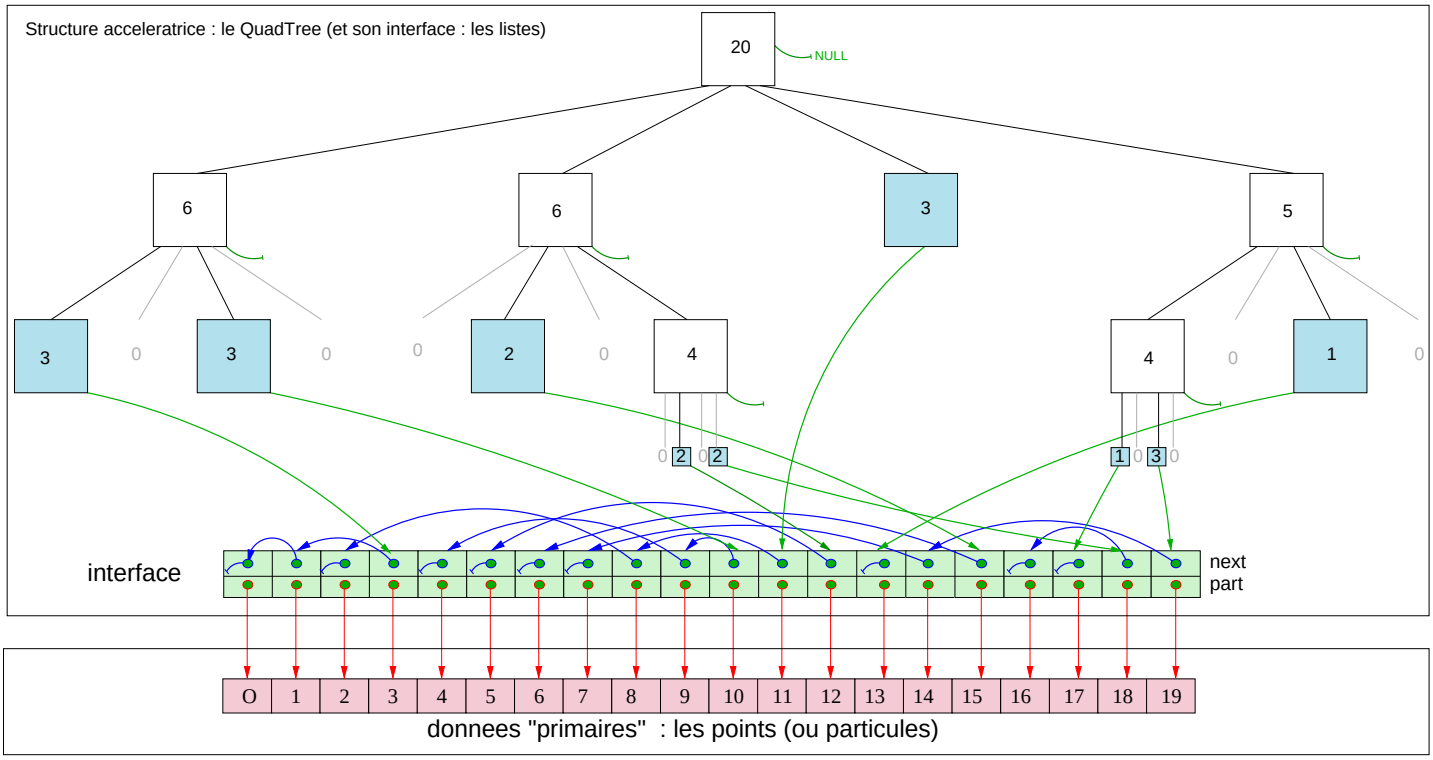
La mise en oeuvre de cette étape n'est pas simple, c'est pour ça qu'on ne l'aborde pas ici.

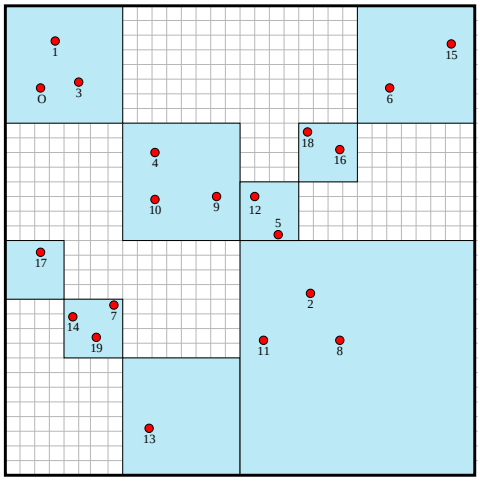
Exemple : dans l'image ci-contre, les données sont

- dimension $W = 2^5 = 32$ et $w_{min} = 1$
- $N_p = 20$ et $K_p = 3$

Et dans le graphique suivant, on trouve le tableau des points et la structure accélératrice complète, y compris le chaînage d'interface.
Les `noeuds` bleus sont les feuilles (qui contiennent des points), les blancs sont les noeuds internes.
Les entiers inscrits dans les `noeuds` du QuadTree correspondent aux champs `nbp`.
Les ⟶ partant des `noeuds` représentent les listes de particules
Les ⟶ sur la liste des `cellules` représentent le chaînage des particules



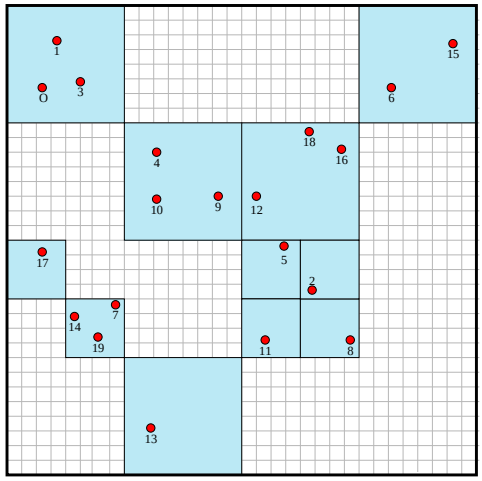




dans cet exemple, il suffirait que le point n°5 se déplace légérement vers le bas pour que la configuration du QuadTree passe de

⟵ ça

à ça ⟶



Mise en oeuvre

① **initialisation du** QuadTree :

La seule information nécessaire pour cela est la taille de la fenêtre de travail $W = 2^h$. On rappelle que le QuadTree est construit dés le départ de manière à couvrir l'intégralité de la zone jusqu'au pixel.

② **distribution des points** :

Comme dans le projet du 1er semestre, votre programme devra proposer deux façons de distribuer les points : de manière aléatoire et un par un au clic souris.

③ **mise à jour du** QuadTree : intégration des points, chaînage, affichage

Pour cette opération, la plus importante, il faut avoir fixé les paramètres annexes K_p et w_{min}.

Les points sont insérés dans le QuadTree séquentiellement, à partir de la racine : tant que, pour le `noeud` courant, la limite K_p n'est pas atteinte, les points s'accumulent sur ce `noeud` (i.e. dans sa liste). Au $(K_p + 1)^{\circ}$ point, la procédure de **purge récursive** se déclenche et commence à segmenter l'espace.

Si le `noeud` courant est tel que `a` → `nbp` > K_p c'est qu'il a déjà été purgé. On descend donc directement dans les fils.

Pour déterminer sur quel `fils` chaque point doit être redistribué, il suffit de quelques considérations géométriques simples (les `noeuds` contiennent les informations nécéssaires – et puisqu'on travaille dans des carrés, on pourra utiliser la "norme infinie" : $\text{dist}_\infty(A, B) = \max(|x_B - x_A|, |y_B - y_A|)$).

- distribution au clic-souris : on ajoute un point, on met à jour le QuadTree et on affiche.
- distribution aléatoire : on traite tous les points puis on affiche le résutat final, ou on produit une "animation" (si le nombre de points est raisonnable) pour voir l'évolution du QuadTree.

☞ vous avez déjà fait tout ça !

L'interactivité et la qualité graphique de votre programme doit mettre en valeur la pertinence de vos réalisation.

Conclusion

Ici, comme souvent, plusieurs types de structures de données cohabitent : tableaux, listes et arbres. Il est important, selon l'opération à effectuer de choisir la structure la plus adéquate.

Et même si dans ce petit devoir nous n'irons pas jusqu'au bout, c'est-à-dire jusqu'à exploiter le QuadTree pour réduire la complexité d'un problème en $o(N^2)$, il faut garder cet objectif à l'esprit.

Mettre en place une *«structure accélératrice»* à toujours un coût (en mémoire, en calcul). Il est indispensable que le bilan reste positif (*«Le mieux est l'ennemi du bien»* – Voltaire, 1772)

Concrètement, un tel modèle ne prend son intérêt que pour des valeurs de N assez grandes. Pour quelques dizaines de points il est disproportionné et contre-productif.