

Struktury QuadTree i Kd-drzewa

Maciej Kmąk Michał Szymocha

Wydział Informatyki AGH

Plan prezentacji



- 1 Przedstawienie problemu
- 2 Drzewa czwórkowe QuadTree
- 3 Kd-drzewa
- 4 Porównanie struktur

Problem wyszukiwania punktów w obszarze



Opis problemu:

- Dane są punkty $P_1 = (x_1, y_1)$ oraz $P_2 = (x_2, y_2)$, które definiują prostokąt w przestrzeni dwuwymiarowej.
- Zbiór P to punkty w przestrzeni dwuwymiarowej.
- Zadanie polega na znalezieniu wszystkich punktów $Q \subseteq P$, dla których:

$$x_1 < x_Q < x_2 \quad \text{oraz} \quad y_1 < y_Q < y_2.$$

Problem wyszukiwania punktów w obszarze



Opis problemu:

- Dane są punkty $P_1 = (x_1, y_1)$ oraz $P_2 = (x_2, y_2)$, które definiują prostokąt w przestrzeni dwuwymiarowej.
- Zbiór P to punkty w przestrzeni dwuwymiarowej.
- Zadanie polega na znalezieniu wszystkich punktów $Q \subseteq P$, dla których:

$$x_1 < x_Q < x_2 \quad \text{oraz} \quad y_1 < y_Q < y_2.$$

Cel projektu:

- Zaimplementowanie struktur danych QuadTree i KdTree.
Struktury mają umożliwiać szybkie odpowiadanie na zapytania o punkty znajdujące się w zadanym prostokątnym obszarze.
- Porównanie wydajności zaimplementowanych struktur.

Propozycje rozwiązania

Możemy odpowiedzieć na to pytanie, przechodząc po całym zbiorze punktów, sprawdzając warunek dla każdego z nich lub skorzystać z bardziej zaawansowanych struktur, takich jak: QuadTree i Kd-Drzewa

Propozycje rozwiązania

Możemy odpowiedzieć na to pytanie, przechodząc po całym zbiorze punktów, sprawdzając warunek dla każdego z nich lub skorzystać z bardziej zaawansowanych struktur, takich jak: QuadTree i Kd-Drzewa

Porównanie algorytmów:

- **Algorytm podstawowy – rozwiązanie naiwne:**
 - Złożoność czasowa: $O(n)$
- QuadTree:
 - Złożoność czasowa: $O(\log n + k)$
- Kd-Drzewa:
 - Złożoność czasowa: $O(\sqrt{n} + k)$

n – liczba punktów zbioru P

k – liczba punktów wynikowych

Rozwiążanie naiwne



Sprawdzamy każdy punkt, takie podejście ma złożoność czasową $O(n)$.

Rozwiążanie naiwne



Sprawdzamy każdy punkt, takie podejście ma złożoność czasową $O(n)$.

Dla prostokąta $R = [x_1, x_2] \times [y_1, y_2]$:

```
filter(lambda p: x_1 <= p[0] <= x_2 and y_1 <= p[1] <= y_2, P)
```

Struktura danych QuadTree

QuadTree to struktura danych umożliwiająca przechowywanie punktów w przestrzeni dwuwymiarowej. Jej głównym zadaniem jest rekurencyjny podział przestrzeni na mniejsze części poprzez dzielenie jej na cztery równe ćwiartki.

Struktura danych QuadTree

QuadTree to struktura danych umożliwiająca przechowywanie punktów w przestrzeni dwuwymiarowej. Jej głównym zadaniem jest rekurencyjny podział przestrzeni na mniejsze części poprzez dzielenie jej na cztery równe ćwiartki.

- Każda z tych ćwiartek może być dalej dzielona na kolejne cztery części, dopóki liczba punktów w danym obszarze nie przekroczy zdefiniowanej maksymalnej pojemności węzła `max_capacity`.

Struktura danych QuadTree

QuadTree to struktura danych umożliwiająca przechowywanie punktów w przestrzeni dwuwymiarowej. Jej głównym zadaniem jest rekurencyjny podział przestrzeni na mniejsze części poprzez dzielenie jej na cztery równe ćwiartki.

- Każda z tych ćwiartek może być dalej dzielona na kolejne cztery części, dopóki liczba punktów w danym obszarze nie przekroczy zdefiniowanej maksymalnej pojemności węzła `max_capacity`.
- Węzły drzewa reprezentują prostokątne obszary przestrzeni.

Struktura danych QuadTree

QuadTree to struktura danych umożliwiająca przechowywanie punktów w przestrzeni dwuwymiarowej. Jej głównym zadaniem jest rekurencyjny podział przestrzeni na mniejsze części poprzez dzielenie jej na cztery równe ćwiartki.

- Każda z tych ćwiartek może być dalej dzielona na kolejne cztery części, dopóki liczba punktów w danym obszarze nie przekroczy zdefiniowanej maksymalnej pojemności węzła `max_capacity`.
- Węzły drzewa reprezentują prostokątne obszary przestrzeni.
- Liście to najmniejsze prostokąty, które nie są już dzielone z powodu ograniczenia liczby punktów.

QuadTree – teoretyczna złożoność



- **Budowa struktury:** $O(n \log n)$, gdzie n to liczba punktów w zbiorze, przy założeniu równomiernego rozkładu punktów.

QuadTree – teoretyczna złożoność

- **Budowa struktury:** $O(n \log n)$, gdzie n to liczba punktów w zbiorze, przy założeniu równomiernego rozkładu punktów.
- **Wyszukiwanie punktów w zadanym obszarze:** $O(\log n + k)$, gdzie:
 - $\log n$ to maksymalna głębokość drzewa w dobrze zrównoważonej strukturze,
 - k to liczba punktów wewnątrz zadanego obszaru.

Dzięki rekurencyjnemu podziałowi przestrzeni wyszukiwanie jest ograniczone tylko do podobszarów przecinających zadany obszar, co znacząco poprawia efektywność.

Budowa QuadTree (opis kroków)

Krok 1: Obliczenie prostokąta obejmującego punkty

Wyznacz prostokąt graniczny, który obejmuje wszystkie punkty wejściowe. Ustaw go jako korzeń drzewa.

Krok 2: Sprawdzenie liczby punktów

Jeśli liczba unikalnych punktów w prostokącie jest mniejsza lub równa maks_pojemność, zakończ podział. Węzeł staje się liściem i przechowuje punkty.

Krok 3: Podział prostokąta na ćwiartki

Wyznacz środek prostokąta jako płaszczyzny podziału i podziel punkty na cztery podzbiory: Dół-lewo (SW), Dół-prawo (SE), Góra-lewo (NW), Góra-prawo (NE).

Budowa QuadTree (opis kroków) – cd.



Krok 4: Rekurencyjny podział

Dla każdej ćwiartki utwórz węzeł podrzędny, przypisz do niego punkty należące do tej ćwiartki, i powtarzaj proces, dopóki liczba punktów w węźle przekracza maks_pojemność.

Krok 5: Zakończenie

Węzły zawierające maks_pojemność punktów lub mniej stają się liśćmi, a drzewo zostaje zbudowane.

QuadTree — Wizualizacja budowy



Wyszukiwanie w QuadTree (opis kroków)

Krok 1: Sprawdzenie kolizji prostokątów

Sprawdź, czy prostokąt zapytania przecina prostokąt bieżącego węzła.

Krok 2: Przeszukiwanie liścia

Jeśli węzeł jest liściem, przeszukaj punkty wewnątrz i zwróć te, które należą do prostokąta zapytania.

Krok 3: Rekurencyjne przeszukiwanie dzieci

Jeśli węzeł nie jest liściem, przeszukaj wszystkie jego dzieci, które mają wspólny obszar z prostokątem zapytania.

Krok 4: Zwrócenie wyników

Zwróć znalezione punkty lub ich licznosci w prostokącie zapytania.

QuadTree — Wizualizacja przeszukiwania



QuadTree – zastosowania



QuadTree jest strukturą używaną głównie w informatyce graficznej, przetwarzaniu obrazu, analizie przestrzennej oraz w problemach związanych z organizacją danych przestrzennych.

QuadTree – zastosowania

QuadTree jest strukturą używaną głównie w informatyce graficznej, przetwarzaniu obrazu, analizie przestrzennej oraz w problemach związanych z organizacją danych przestrzennych.

Przykłady zastosowania struktury:

- Kompresja obrazu
- Segmentacja obrazu
- Renderowanie terenu
- Klipping (proces eliminacji obiektów, które nie mieszczą się w określonym obszarze widoczności)
- Wyszukiwanie punktów
- Przyspieszanie zapytań przestrzennych
- Zarządzanie kolizjami
- Klasterowanie danych

QuadTree – Analogiczne Zastosowanie Geodezyjne

Podział stosowany w układach geodezyjnych jest analogiczny do struktury drzewa QuadTree, gdzie każdy większy obszar jest dzielony na cztery mniejsze części. Dzięki temu można efektywnie zarządzać dużymi zbiorami danych przestrzennych, zarówno lokalnych, jak i globalnych.

QuadTree – Analogiczne Zastosowanie Geodezyjne

Podział stosowany w układach geodezyjnych jest analogiczny do struktury drzewa QuadTree, gdzie każdy większy obszar jest dzielony na cztery mniejsze części. Dzięki temu można efektywnie zarządzać dużymi zbiorami danych przestrzennych, zarówno lokalnych, jak i globalnych.

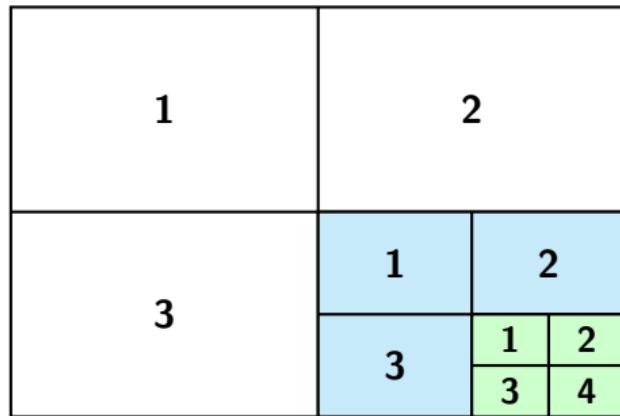
Przykłady zastosowań:

- Polska 1965
- Polska 1992
- Polska 2000
- WGS 84 (system globalny)
- ETRS89 (Europejski Terestrialny System Odniesienia)

QuadTree – Analogiczne Zastosowanie Geodezyjne – cd.

W układzie "1965" każde godło mapy opisuje położenie i podział arkusza w odpowiedniej skali. Separator (kropka) między segmentami godła pozwala łatwo identyfikować podział i skalę mapy. Przykład:

- **1:5 000** → 263.123.2 Oznacza drugą ćwiartkę arkusza 1:10 000 (263.123)
- **1:2 000** → 263.123.24 Oznacza czwartą ćwiartkę drugiej ćwiartki arkusza 1:10 000 (263.123)
- **1:1 000** → 263.123.242 Oznacza drugą ćwiartkę arkusza 1:2 000 (263.123.24)
- **1:500** → 263.123.242.3 Oznacza trzecią ćwiartkę arkusza 1:1 000 (263.123.242)



Ogólna idea kd-drzewo

Dlaczego kd-drzewo?

- kd-drzewo (k-d Tree) to struktura do przechowywania punktów w przestrzeniach wielowymiarowych (u nas 2D).
- Zamiast traktować wszystkie wymiary naraz, dzielimy przestrzeń po jednej osi na danym poziomie drzева (np. raz po x , raz po y).
- Taki rekurencyjny podział sprawia, że do wielu zapytań przestrzennych (np. *czy punkt A leży w danym obszarze?*) nie musimy przeglądać wszystkich danych, a tylko część drzева.
- Dzięki temu kd-drzewo często znacznie przyspiesza wyszukiwanie w porównaniu z metodami naiwnymi, zwłaszcza przy dużych zbiorach punktów.

kd-drzewo – teoretyczna złożoność



- **Budowa struktury:** $\Theta(n \log n)$, gdzie n to liczba punktów w zbiorze, przy założeniu równomiernego rozkładu punktów.

kd-drzewo – teoretyczna złożoność

- **Budowa struktury:** $\Theta(n \log n)$, gdzie n to liczba punktów w zbiorze, przy założeniu równomiernego rozkładu punktów.

- **Wyszukiwanie punktów w zadanym obszarze:** $\Theta(\sqrt{n} + k)$, gdzie:
 - \sqrt{n} to liczba węzłów jakie odwiedzi algorytm przy założeniu równomiernego rozkładu,
 - k to liczba punktów wewnątrz zadanego obszaru.

Dzięki rekurencyjnemu podziałowi przestrzeni wyszukiwanie jest ograniczone tylko do podobszarów przecinających zadany obszar, co znacząco poprawia efektywność.

Budowa kd-drzewa (opis kroków)

Krok 1: Wybór osi podziału

Na podstawie aktualnego poziomu drzewa (*głębokości*) decydujemy, czy dzielimy względem osi x , czy y . - Jeśli głębokość jest parzysta (0, 2, 4, ...), to dzielimy względem x . - Jeśli nieparzysta (1, 3, 5, ...), to dzielimy względem y .

Krok 2: Wybór punktu podziału (medianą)

Zbieramy listę punktów i sortujemy je według współrzędnej, która odpowiada wybranej osi. Następnie *medianą* (środkowy punkt w posortowanej liście) zostaje węzłem głównym naszego poddrzewa.

Dzięki temu drzewo jest w miarę zbalansowane, bo po obu stronach mediany jest podobna liczba punktów.

Budowa kd-drzewa (opis kroków) – c.d.

Krok 3: Podział zbioru punktów

Wszystkie punkty, które w wybranym wymiarze są „mniejsze” od mediany, przydzielamy do lewego poddrzewa, a punkty „większe” – do prawego poddrzewa. Jeżeli os to x , to dzielimy: $(x \leq x_{median}) \rightarrow$ lewa strona, a $(x > x_{median}) \rightarrow$ prawa strona.

Krok 4: Rekurencja

Powyzsze czynności powtarzamy dla „lewego” i „prawego” zbioru – znów sprawdzając kolejną oś (naprzemiennie). Proces kończy się, kiedy w danym węźle pozostanie już pojedynczy punkt lub zbiór będzie pusty.

Wizualizacja budowy kd-drzewa



Wyszukiwanie punktów w obszarze $[x_1, x_2] \times [y_1, y_2]$

Krok 1: Sprawdzenie punktu w aktualnym węźle

Rozpoczynamy od korzenia drzewa i sprawdzamy, czy punkt w tym węźle (zапам'ятана mediana) mieści się w obszarze $[x_1, x_2] \times [y_1, y_2]$. Jeżeli tak – dodajemy go do wyniku.

Krok 2: Analiza podziału drzewa

Patrzymy, w którą stronę naszego obszaru *mogą* spadać interesujące nas punkty.

- Jeśli dzielimy względem x i nasza lewa granica (x_1) jest mniejsza bądź równa od współrzędnej x bieżącego punktu, to **wiemy**, że w lewym poddrzewie mogą być punkty w zakresie.
- Z kolei, jeśli prawa granica (x_2) jest większa bądź równa od tej współrzędnej, to idziemy też w prawo.
- Analogicznie postępujemy, gdy podział jest względem y .

Wyszukiwanie punktów w obszarze $[x_1, x_2] \times [y_1, y_2]$ – c.d.



Krok 3: Rekurencja

Przechodzimy do węzłów lewego i/lub prawego, powtarzając ten sam schemat:

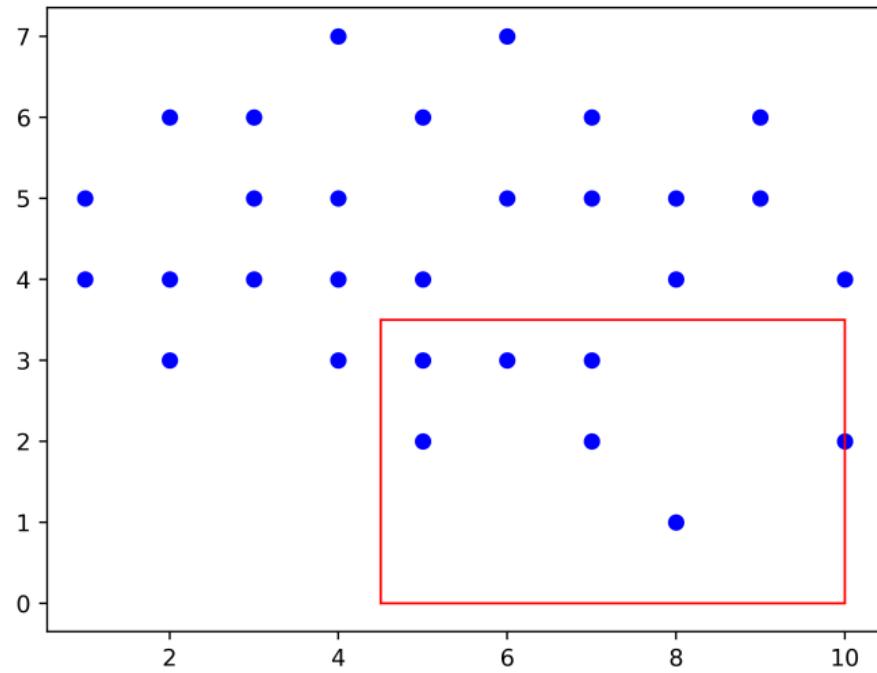
- Czy punkt węzła mieści się w zakresie? - Czy warto iść dalej w lewo/prawo, biorąc pod uwagę porównanie współrzędnej węzła z $[x_1, x_2]$ albo $[y_1, y_2]$?

Dzięki temu „omijamy” poddrzewa, które na pewno nie mają punktów w naszym obszarze.

Krok 4: Zakończenie wyszukiwania

Kiedy dotrzymy do pustych węzłów lub przejdziemy przez całe drzewo w miejsca potencjalnie zawierające punkty w zakresie, kończymy rekurencję. Wszystkie punktu zebrane po drodze *muszą* być tymi, które leżą w zadanym prostokącie $[x_1, x_2] \times [y_1, y_2]$.

Wizualizacja zapytania dla kd-drzewa



Wizualizacja zapytania dla kd-drzewa



Budowa kd-drzewa w Pythonie i złożoność

- **Dlaczego to może być $O(n \log^2 n)$?**

- Na każdym poziomie drzewa sortujemy listę punktów ($O(n \log n)$), a poziomów jest $\log n$ w przypadku dość równomiernego podziału.
- Sumarycznie: $O(n \log n) \times \log n = O(n \log^2 n)$.

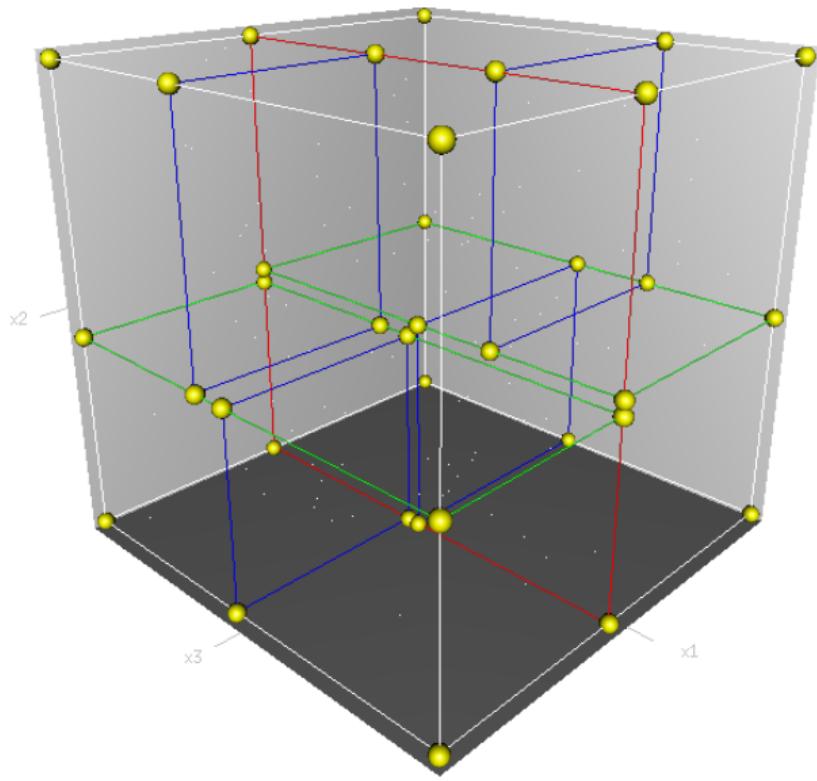
- **Czy da się osiągnąć $O(n \log n)$?**

- Tak, np. używając szybkich algorytmów wyszukiwania mediany (Quickselect, Introselect) zamiast pełnego sortowania na każdym poziomie.
- Jednak w praktyce implementacja Quickselect/Introselect bywa bardziej skomplikowana i w gorszym przypadku może się źle skalować (np. Quickselect ma pesymistycznie $O(n^2)$).
- Często prostota kodu (sortowanie) bywa ważniejsza niż teoretyczny zysk, zwłaszcza przy realnych danych.

Zastosowania kd-drzewa

- **Wyszukiwanie najbliższych sąsiadów (ang. Nearest Neighbor Search):**
 - k-NN w uczeniu maszynowym (np. klasyfikacja, regresja).
 - Znajdowanie najbardziej podobnych obiektów w dużych zbiorach danych.
- **Zapytania przestrzenne:**
 - Wyszukiwanie punktów w zadanym obszarze (prostokącie, kuli itp.).
 - Optymalizacja zapytań GIS, np. w bazach danych geograficznych.
- **Klasteryzacja i analiza danych wielowymiarowych:**
 - Skupiska punktów w 2D lub wyższych wymiarach.
 - Przyspieszanie algorytmów typu DBSCAN, Mean Shift itp.
- **Wizualizacja i grafika komputerowa:**
 - Struktura przyspieszająca techniki renderowania (np. *ray tracing*).
 - Wspomaganie detekcji kolizji w symulacjach i grach 2D/3D.
- **Systemy rzeczywiste:**
 - Aplikacje robotyczne (wyznaczanie najbliższych przeszkód).
 - Motory rekomendujące (wielowymiarowe porównywanie cech produktów).

Generalizacja do k-wymiarów



kd-drzewo vs QuadTree – Różnice w strukturze

- **Podział przestrzeni:**

- kd-drzewo: na każdym poziomie dzielimy przestrzeń względem jednej osi (x lub y), naprzemiennie.
- QuadTree: na każdym poziomie dzielimy przestrzeń na cztery równe ćwiartki (osie x i y jednocześnie).

- **Układ węzłów drzewa:**

- kd-drzewo: węzeł przechowuje pojedynczy punkt (mediana dla danej osi). Dzieli się na *lewe* i *prawe* poddrzewo.
- QuadTree: węzeł może przechowywać wiele punktów (do pewnego limitu), potem następuje podział zwykły na cztery poddrzewa (*NW*, *NE*, *SW*, *SE*).

- **Wielowymiarowość:**

- kd-drzewo: naturalnie rozszerza się na k wymiarów (stąd nazwa).
- QuadTree: typowo używane w 2D, choć istnieją analogie (tzw. OctTree w 3D).

kd-drzewo vs QuadTree – Złożoność i zastosowania

Złożoność i efektywność

- **Budowa:**

- kd-drzewo: często $O(n \log n)$ (przy równomiernym rozkładzie i efektywnym wybieraniu median).
- QuadTree: $O(n \log n)$ przy równomiernym rozkładzie, choć implementacja oparta jest na iteracyjnym podziale przestrzeni, a nie na wyszukiwaniu median.

- **Wyszukiwanie:**

- kd-drzewo: Średnio $O(\sqrt{n} + k)$ w 2D, gdzie k to liczba punktów w obszarze.
- QuadTree: Także średnio $O(\log n + k)$, przy założeniu równomiernego rozkładu (a w praktyce bywa różnie).

Kiedy wybrać kd-drzewo a kiedy QuadTree?

Podsumowanie kryteriów wyboru:

- $k \geq 3$: → kd-drzewo naturalnie się rozszerza na k wymiarów, QuadTree (2D) nie. (OctTree jest 3D analogią Quadtree, ale bywa bardziej złożony w implementacji.)
- **Wyszukiwanie w obszarze vs. Wyszukiwanie najbliższych sąsiadów:**
 - kd-drzewo jest często lepsze do najbliższych sąsiadów, szczególnie w 2D/3D.
 - QuadTree wystarczająco dobrze radzi sobie z przeszukiwaniem obszarów, szczególnie przy równomiernym rozkładzie w 2D.
- **Implementacja i prostota:**
 - QuadTree – dość prosty kod (rekurencyjny podział na 4 części).
 - kd-drzewo – wymaga wyznaczania mediany (lub sortowania) przy budowie, ale jest dość uniwersalny w wymiarach > 2 .

Zastosowania

Typowe zastosowania

- kd-drzewo:
 - Wyszukiwanie najbliższych sąsiadów (*ang. Nearest Neighbor Search*).
 - Klasteryzacja w przestrzeniach wielowymiarowych.
 - Zapytania typu przeszukiwanie obszaru (dla 2D, 3D i więcej wymiarów).
- QuadTree:
 - Zadania stricte 2D: grafika komputerowa, operacje na obrazach, GIS.
 - Renderowanie terenu, kolizje w grach 2D.
 - Kompresja i segmentacja obrazów (*przetwarzanie obrazów*).

Zobaczmy jak to wygląda w kodzie

Koniec. Dziękujemy za uwagę!