Tomasz Cudek
Marcin Paśko
IV rok, Informatyka

Implementacja algorytmu triangulacji zbioru punktów z wykorzystaniem struktury quadtree

Opis algorytmu

Przy implementacji metody oparliśmy się na artykule „*Provably Good Mesh Generation*" [1] autorstwa Marshalla Berna, Davida Eppsteina oraz Johna Gilberta. Program implementuje najbardziej podstawową wersję metody generacji siatki trójkątnej dla zbioru punktów opisaną w drugim rozdziale artykułu, metoda ta stanowi bazę do kolejnych optymalizacji pozwalających w efekcie końcowym uzyskać bardzo dobrą siatkę optymalizującą zarówno kształt trójkątów jak i ich liczbę (nie zaimplementowane). Algorytm triangulacji utworzony został z myślą o generacji siatki dobrze uwarunkowanej na potrzeby obliczeń wykorzystujących metodę elementów skończonych, zatem siatka zwracana przez nasz program szczególnie nadaje się do tego typu zastosowań.

Zaproponowana metoda generuje najpierw drzewo QuadTree, nad zbiorem punktów, po czym wykonuje na nim kolejne transformacje, aż do momentu gdy jego struktura spełniać będzie określone kryteria, po czym następuje triangulacja. Realizowany algorytm daje współczynnik proporcjonalności (*aspekt ratio)* mniejszy niż 4[1].

Definicje

Kwadrat *(box)* – jest to węzeł drzewa quadtree

Podzielony kwadrat (*Split)* – kwadrat niebędący liściem, zawierający cztery dzieci

Sąsiad kwadratu - przez sąsiada kwadratu rozumiemy kwadrat graniczący z nim na jednym z boków, będący tego samego rozmiaru co dany kwadrat. W dalszym części tekstu uogólniamy pojęcie sąsiada na kwadrat dowolnej wielkości.

Rozszerzony sąsiad kwadratu - przez rozszerzonego sąsiada kwadratu rozumiemy kwadrat graniczący z danym kwadratem poprzez współdzielenie wierzchołka lub boku.

Kwadrat zatłoczony (*crowded box*) - przez zatłoczony kwadrat rozumiemy:

- kwadrat posiadający więcej niż jeden punkt
- kwadrat zawierający punkt, którego odległość od najbliższego innego punktu jest mniejsza od $2\sqrt{2}l$, gdzie przez *l* rozumiemy długość boku kwadratu
- kwadrat zawierający punkt i posiadający przynajmniej jednego rozszerzonego sąsiada będącego kwadratem podzielonym

Zatłoczone drzewo quadtree – drzewo w którym przynajmniej jeden z węzłów jest zatłoczony

Drzewo zrównoważone – przez zrównoważone drzewo quadtree rozumiemy drzewo, w którym nie istnieje taki kwadrat, że dowolny z jego sąsiadów jest od niego ponad dwa razy większy.

Założenia

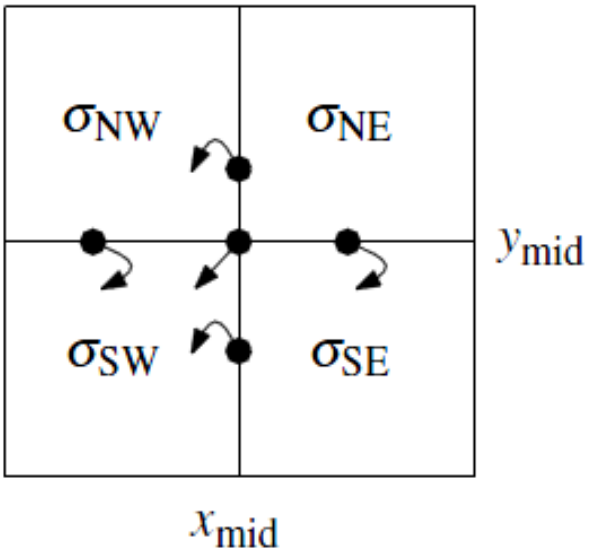
W zaimplementowanej metodzie, by można było przystąpić do triangulacji drzewa QuadTree wygenerowanego nad zbiorem punktów dla którego tworzymy siatkę, musi być ono poprawnie uformowane poprzez spełnienie następujących warunków:

- drzewo nie może być zatłoczone
- drzewo musi być zrównoważone

Ponadto autorzy artykułu sugerują również, by po każdym podziale kwadratu zawierającego punkt otaczać dziecko, w którym znajdzie się punkt po podziale, pierścieniem złożonym z ośmiu rozszerzonych sąsiadów tej samej wielkości, co zostało również uwzględnione w implementacji.

Klasyfikacja punktów leżących na krawędziach quadtree

Punkty krawędziowe klasyfikujemy w sposób zaproponowany w książce *„Computational Geometry: Algorithms and Applications”* [2]:



Klasyfikacja punktów wierzchołkowych i krawędziowych

Implementacja

Algorytm realizujemy korzystając przede wszystkim z obiektu klasy QuadTree, udostępniającego metody do podziałów siatki drzewa. W wewnętrznej strukturze drzewa przechowujemy również zbiór punktów dla którego przeprowadzana jest transformacja siatki. Po zakończeniu podziałów drzewa wywołujemy na nim metodę *triangulate()*, dokonującą triangulacji. Wykonanie całego algorytmu następuje poprzez wywołanie kilku wysoko poziomowych metod:

Wysokopoziomowy kod

Poniżej przedstawiono ciąg wywołań funkcji w mainie, prowadzący do wygenerowania siatki:

```
ofstream* output_stream;
/** wskaźnik na quadtree*/
QuadTree* qt;
/** wskaźnik na zbiór punktów dla którego jest generowana siatka*/
list<Point*>* points_list;

/*
 * ...
 * ...
 * Wczytanie danych z pliku lub generacja danych losowych,
 * inicjalizacja listy punktów
 * ...
 * ...
 */

/* KROK 1
 * Utworzenie korzenia drzewa */
qt = create_initial_box(points_list);

/* KROK 2
 * Utworzenie początkowej siatki poprzez dokładanie do niej kolejnych
 * punktów(dzieląc przy tym odpowiednio quadtree tak by dwa punkty nie
 * leżały w jednym kwadracie*/
qt->init_mesh(points_list);

/* KROK 3
 * Podzielenie kwadratów ze względu na warunek odległości między
 * punktami*/
qt->split_too_close_boxes();

/* KROK 4
 * Otoczenie każdego z kwadratów zawierającego punkt oraz wszystkich jego
 * rodziców, aż do korzenia, sąsiadami tej samej wielkości*/
qt->surround_with_neighbours_ascending();

/* KROK 5
 * Równoważenie drzewa*/
qt->balance_tree();

int size = points_list->size();
MergeTable merge(size * size * 100);

/*KROK 6
 * Ustawienie wskaźników na wierzchołki w każdym z kwadratów w taki
 * sposób, by każdy z wierzchołków quadtree przechowywany był tylko
 * pod jednym adresem w pamięci, wcześniej każdy kwadrat przechowywał
 * lokalną kopię wartości swoich współrzędnych
 */
qt->mergeCorners(&merge);
```

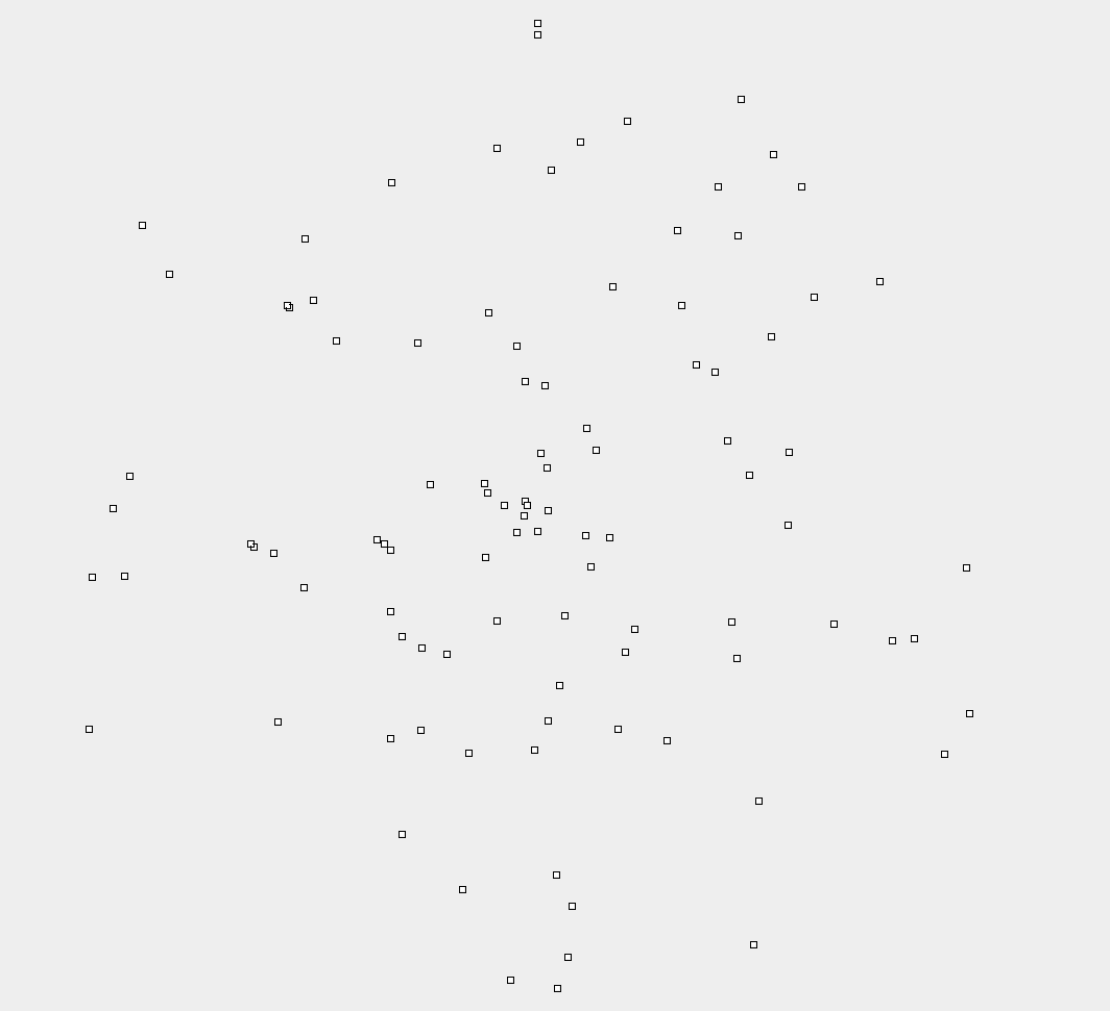
```
/* Naciągnięcie wierzchołków quadtree na punkty dla których generowana
 * jest siatka*/
qt->transform();

/* KROK 7
 * Triangulacja
 */
triangulate(output_stream, qt);
```

Sekwencja instrukcji budująca i transformująca drzewo quadtree oraz generująca siatkę trójkątną dla zbioru punktów

Przegląd poszczególnych kroków algorytmu

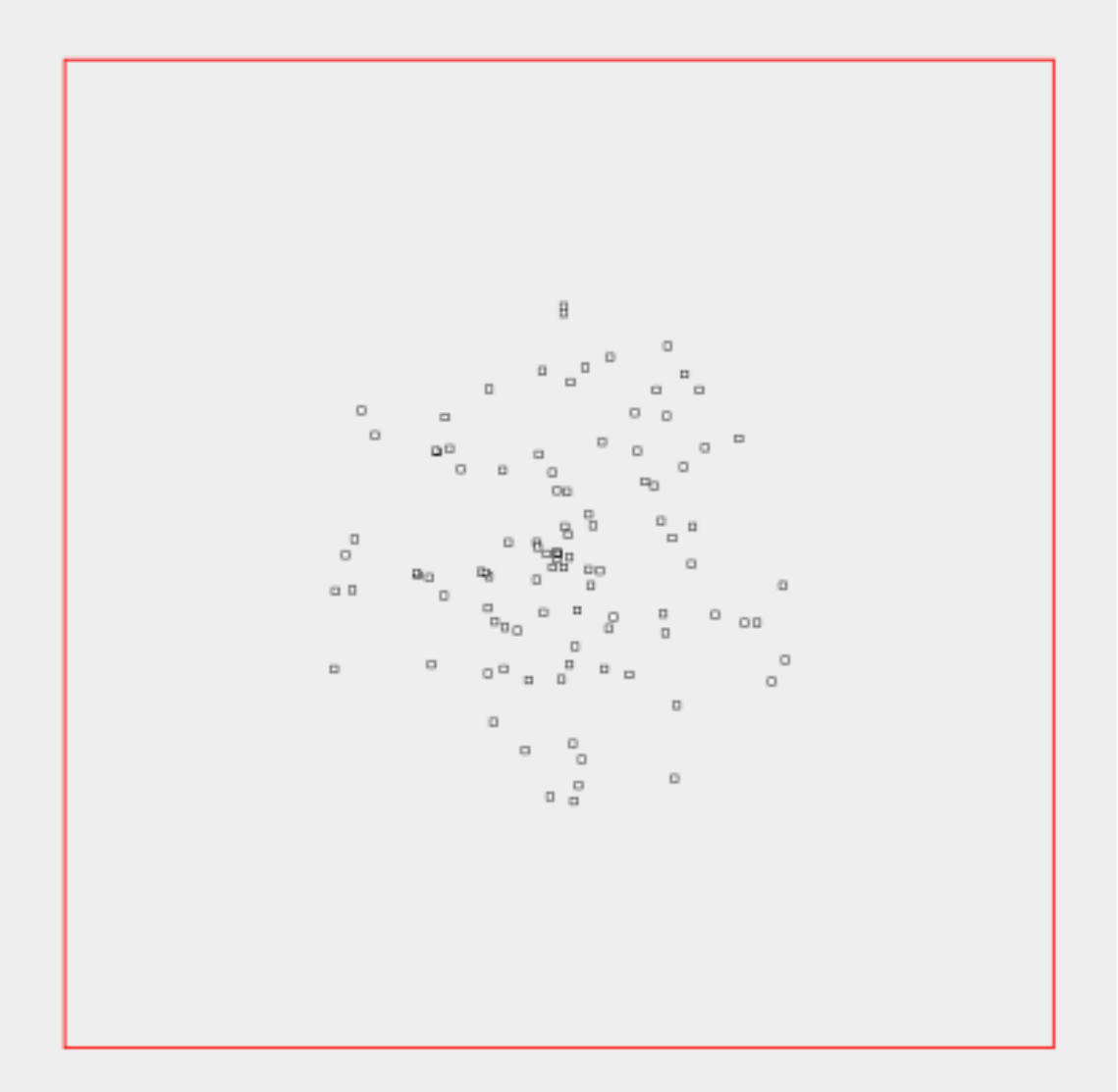
Algorytm rozpatrzymy na przykładzie danych wejściowych będących listą100 punktów wygenerowanych wewnątrz okręgu o promieniu 1000.



Początkowy zbiór punktów

Krok 1 – inicjalizacja struktury QuadTree

Wyznaczenie początkowego kwadratu – korzenia. Środek korzenia zostaje wyznaczony w środku zbioru punktów, jako początkową długość boków kwadratu przyjmujemy podwojoną odległość pomiędzy współrzędnymi x-owymi lub y-owymi (zależnie od tego która jest większa) skrajnych punktów.



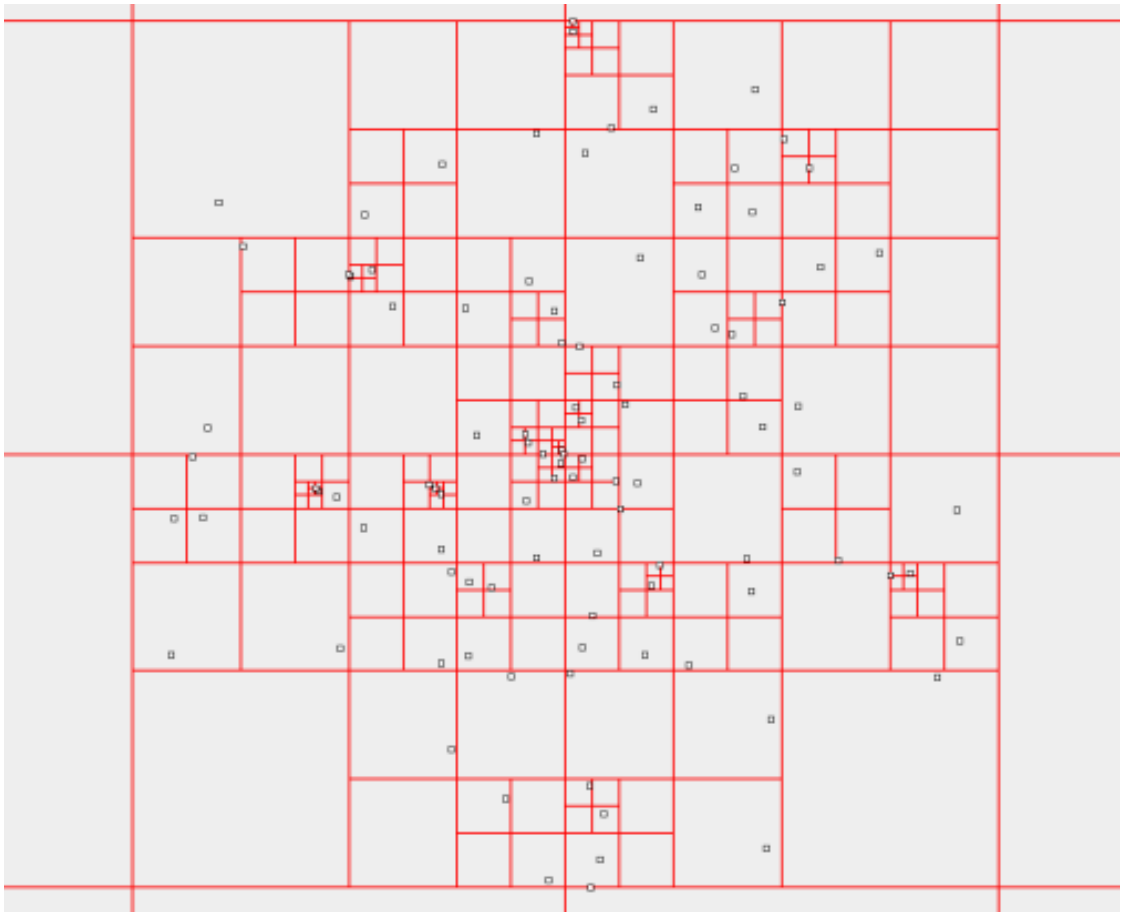
Siatka po pierwszym kroku

Krok 2 – dodawanie punktów do drzewa

Dodajemy do obiektu quadtree po kolei wszystkie punkty dla których generować będziemy siatkę. Sposób dodawania punktów:

Jeżeli węzeł do którego chcemy dodać punkt zawiera już inny punkt węzeł zostaje podzielony, po czym próbujemy dodać punkt po raz kolejny do odpowiedniego dziecka węzła, jeżeli okaże się, że dziecko również zawiera punkt, dzielimy dziecko i tak dalej, do momentu, gdy punkt trafi do osobnego kwadratu.

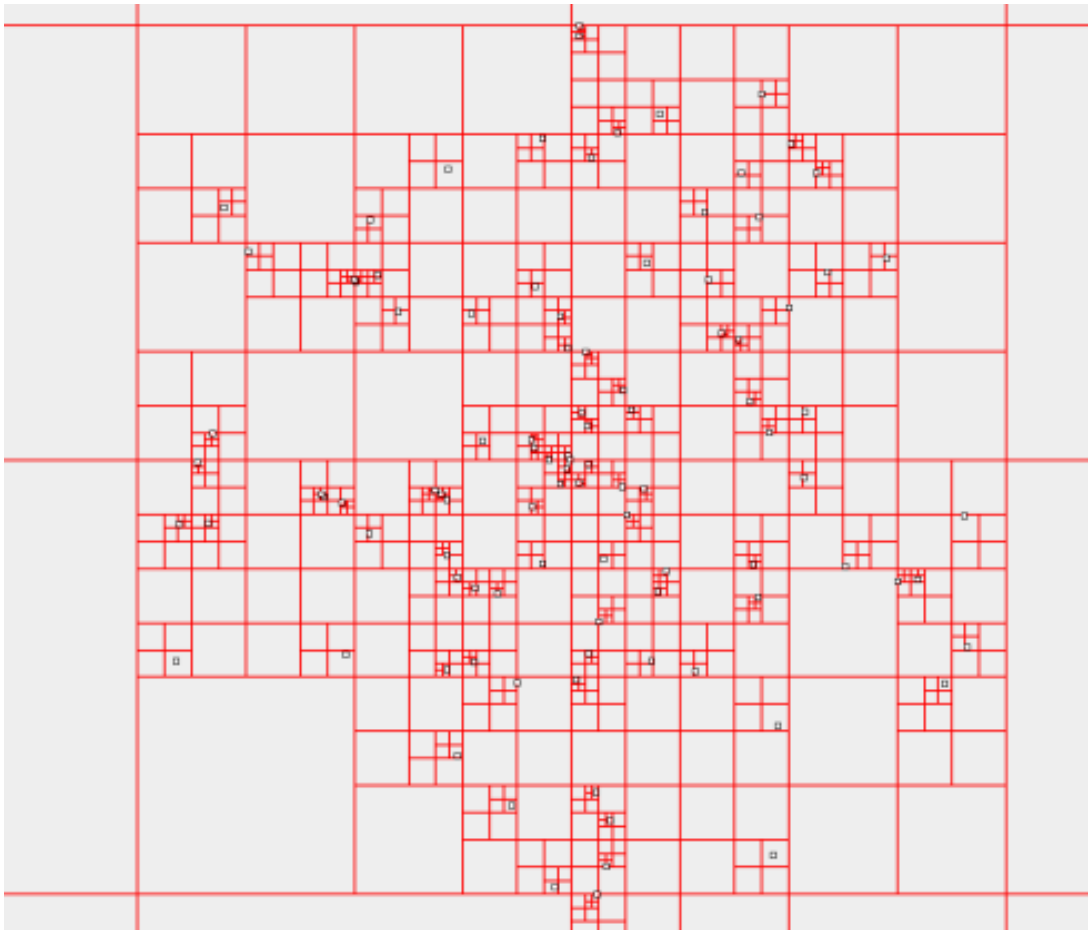
Zatem po wykonaniu tego kroku posiadamy strukturę quadtree, podzieloną w taki sposób, że każdy z punktów znajduje się w osobnym kwadracie. Oznacza to, że wyeliminowany został pierwszy warunek zatłoczenia.



Siatka po drugim kroku – każdy punkt leży w osobnym kwadracie

Krok 3 – podział kwadratów ze względu na odległości między punktami

Dla każdego z punktów wyznaczamy odległość do najbliższego punktu będącego jego sąsiadem, robimy to wyznaczając dla każdego z punktów jego odległość do najbliższego z sąsiadów, w ten sposób możemy dla każdego z punktów z góry obliczyć jaka musi być maksymalna długość boku kwadratu w którym się znajduje, aż przez to wymaganą liczbę podziałów. Dzielmy odpowiednio każdy z kwadratów, struktura którą otrzymamy wyklucza by którykolwiek z kwadratów mógł spełniać drugi warunek zatłoczenia.

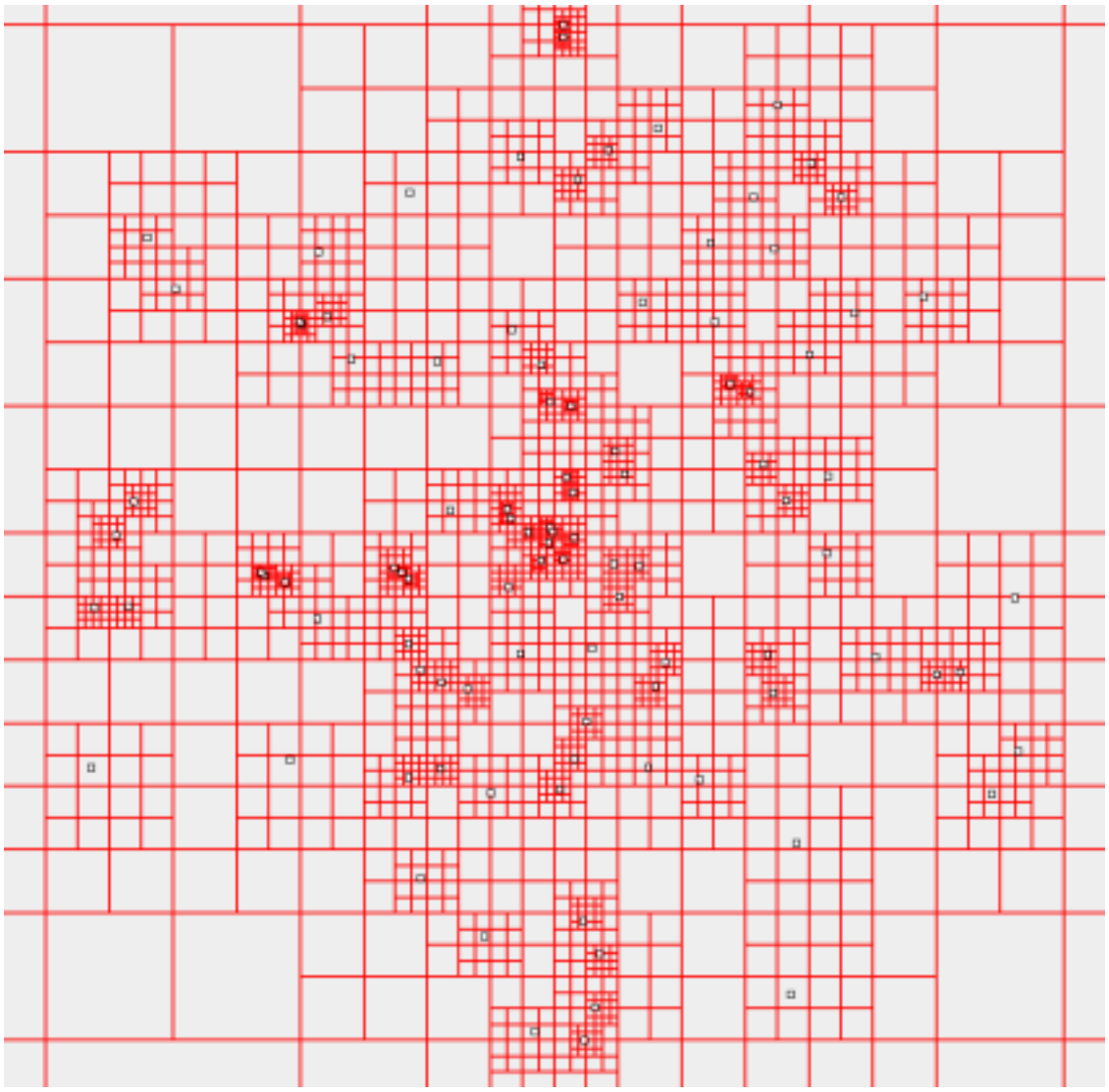


Siatka po trzecim kroku – każdy z punktów leży od pozostałych w odległości większej niż dwie przekątne kwadratu w którym się znajduje

Krok 4 – otoczenie podzielonych kwadratów sąsiadami tego samego rozmiaru

Każdy z kwadratów zawierający punkt otaczamy sąsiadami tego samego rozmiaru. Wykonujemy to samo dla każdego z jego rodziców, aż dojdziemy do korzenia. Krok ten jest wykonywany, ponieważ algorytm wymaga, by po podzielaniu kwadratu zawierającego punkt dziecko tego kwadratu, które zawiera punkt otoczone zostało pierścieniem kwadratów tej samej wielkości. Ponieważ śledzenie, które kwadraty należy dzielić byłoby uciążliwe w poprzednich krokach i wprowadzało niepotrzebny nakład obliczeniowy związany ze śledzeniem propagacji zmian w siatce, realizujemy to jako osobny krok. Otrzymana siatka jest identyczna jak w przypadku, gdybyśmy dzielili ją na bieżąco, ponieważ każdy z rodziców najmniejszego kwadratu z punktem, od którego zaczynamy otaczanie musiałby zostać wcześniej zostać otoczony kwadratami, jako, że musiał być liściem zawierającym punkt. Poza tym żaden inny kwadrat na pewno nie był otaczany, bo otaczanie sąsiadami dotyczy tylko kwadratów zawierających punkty.

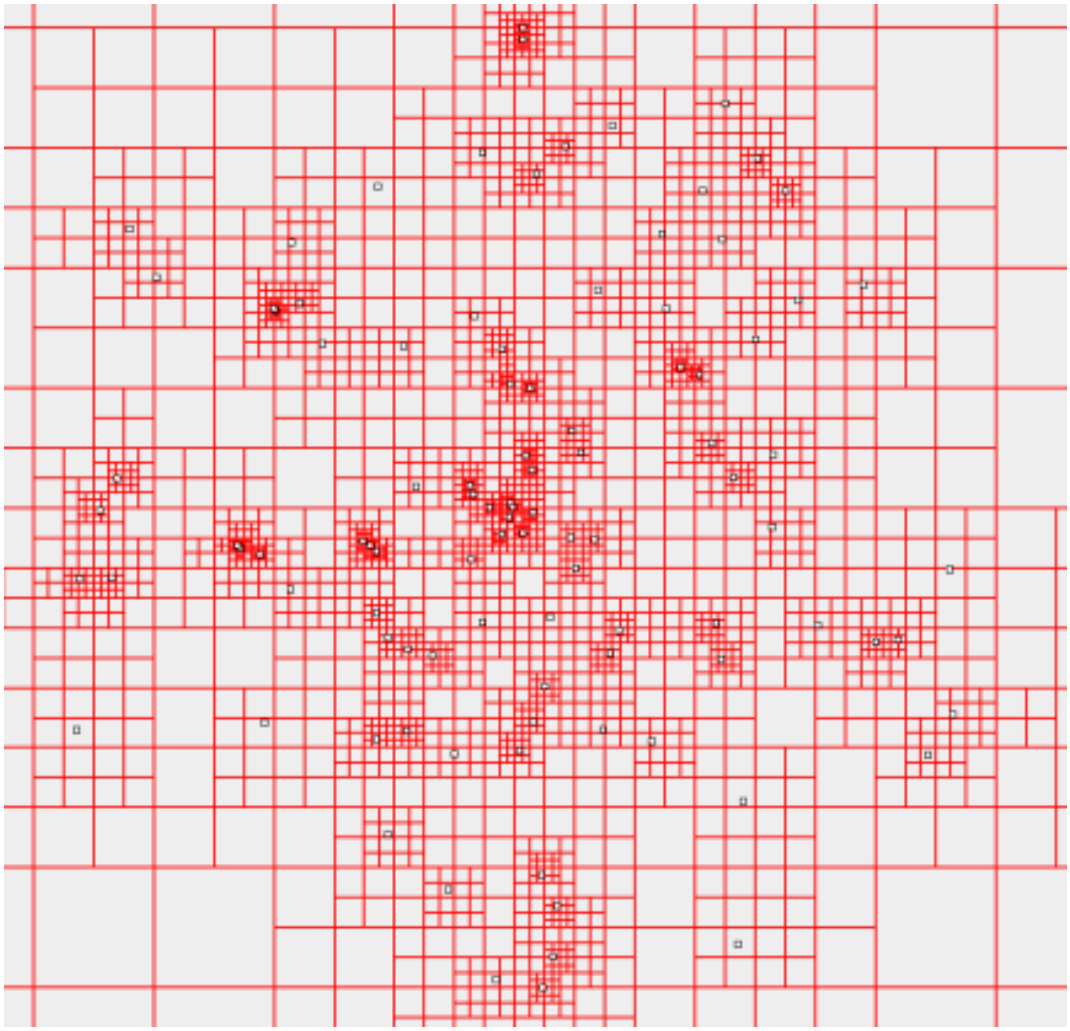
Ponadto możemy zaobserwować, że krok ten spowodował, że spełniony zostaje ostatni z warunków na brak zatłoczenia, jako że żaden z sąsiadów kwadratu z punktem nie jest podzielony.



Siatka po czwartym kroku – każdy z punktów zawierający punkt, oraz jego węzły nadrzędne otoczone zostają pierścieniem sąsiadów

Krok 5 – równoważenie drzewa

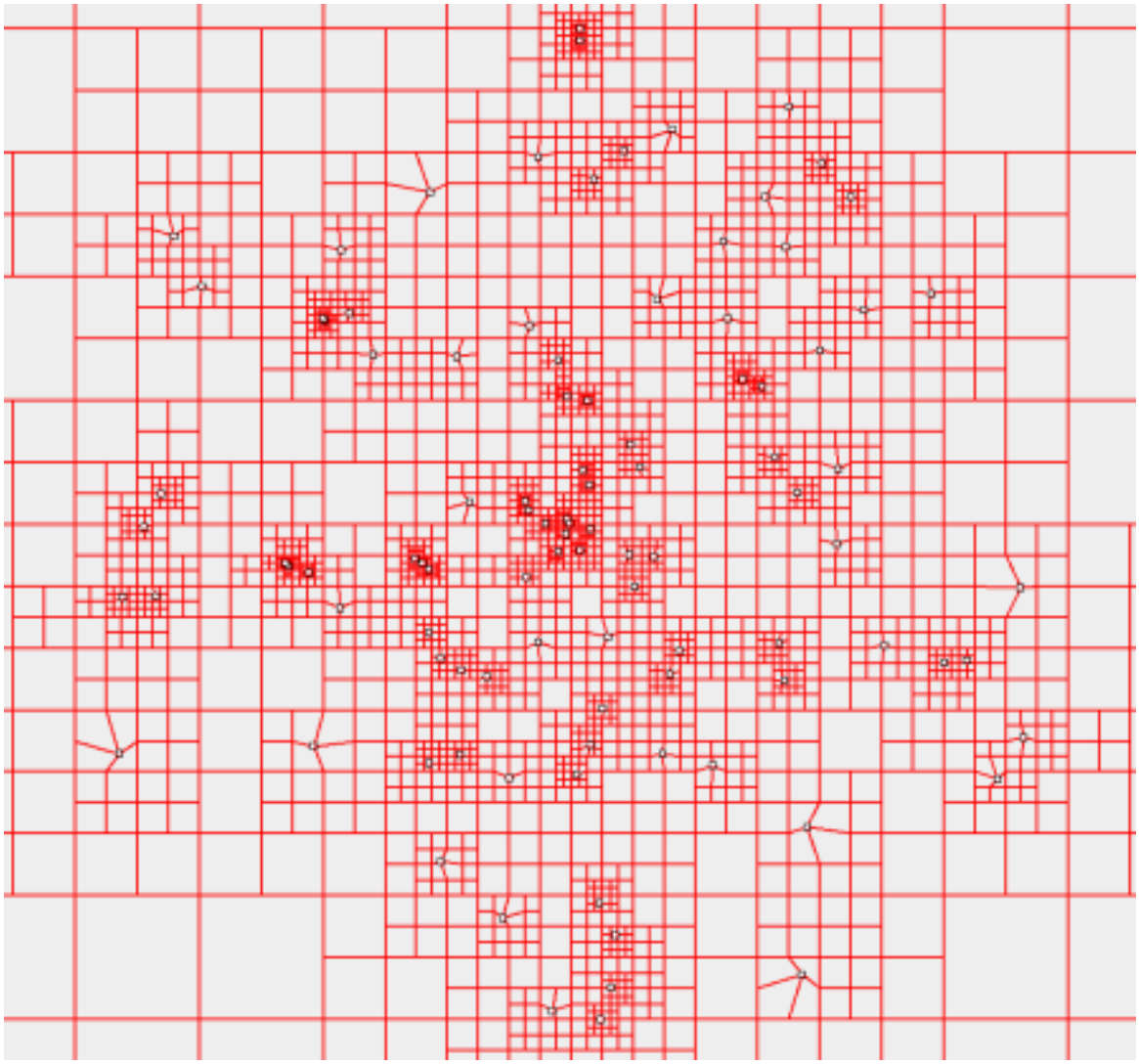
Jest to ostatni z kroków generacji siatki. Równoważenie możemy wykonać na dowolnym etapie algorytmu budowy QuadTree, jako, że operacja ta ma znaczenie tylko dla triangulacji. Równoważenie wykonane jest jako osobny krok ponownie, by nie wprowadzać narzutu związanego ze śledzeniem propagacji zmian siatki w czasie działania algorytmu. Równoważenie wykonywane przez rekurencyjne przejście całego drzewa od korzenia do każdego z liści. Jeżeli warunek równowagi nie jest spełniony w którymś z kwadratów następuje jego podział. Algorytm równoważenia wykonuje się w pętli, która kończy się w momencie, gdy po przejściu całego drzewa żaden z węzłów nie został zmieniony. W praktyce oznacza to zwykle 2 – 3 przejścia.



Siatka po piątym kroku – zrównoważone quadtree

Krok 6 – naciąganie wierzchołków drzewa quadtree (*wrapping*)

Otrzymane do tej pory drzewo nie jest poprawne, pod tym względem, że jego wierzchołki nie pokrywają zbioru punktów, dla których generowana jest siatka. Poprawiamy to poprzez przeciąganie (*wrap*) wierzchołków wygenerowanego quadtree – dla każdego z punktów bierzemy najbliższy z wierzchołków quadtree i zamieniamy jego współrzędne ze współrzędnymi punktu. Wykonanie tego kroku wymaga na początku przestawienia wskaźników na wierzchołki w każdym z kwadratów, w taki sposób, aby każdy wierzchołek drzewa quadtree przechowywany był tylko pod jednym adresem w pamięci (wcześniej każdy kwadrat miał własne kopie współrzędnych swojego wierzchołka). Po wykonaniu kroku otrzymujemy poprawną siatkę, uwzględniającą wejście programu.

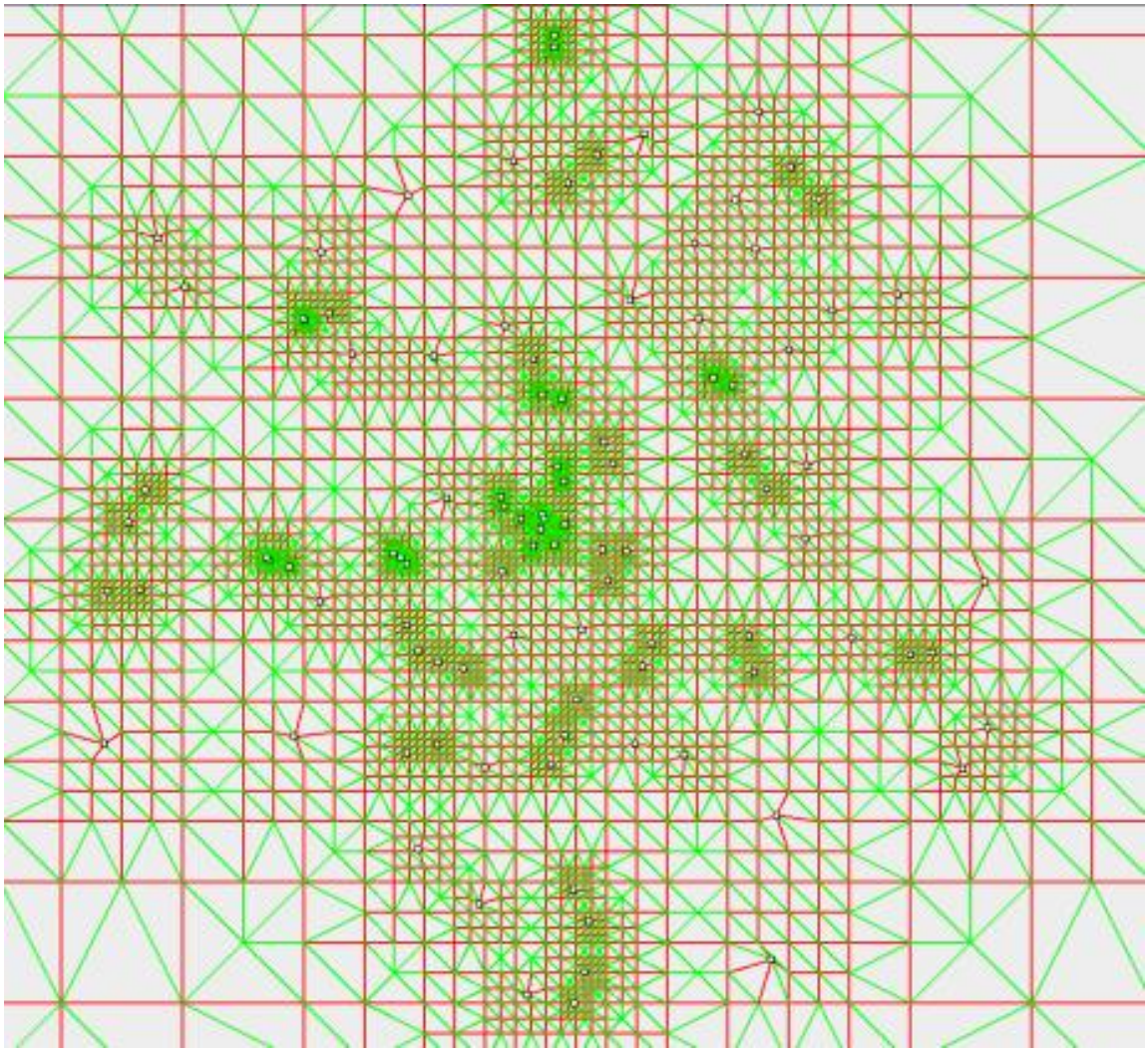


Siatka po szóstym kroku – wierzchołki quadtree zostały dociągnięte do punktów

Krok 7 – triangulacja

Na koniec wykonujemy zwyczajną triangulację Steinera, z małą modyfikacją, która polega na tym, że w momencie, gdy dany kwadrat posiada tylko jeden przecięty bok, to triangulację dla tego kwadratu

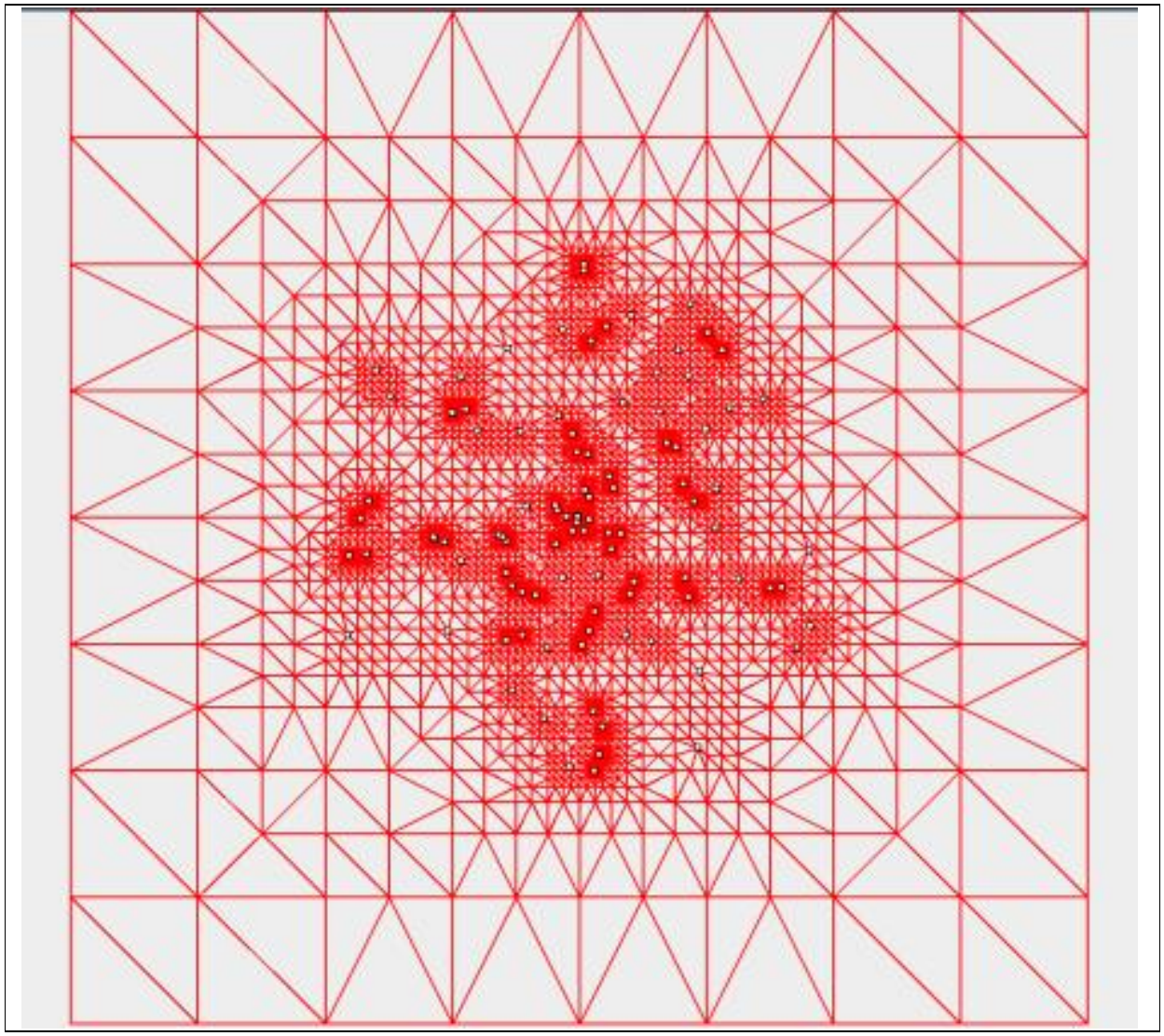
tworzymy poprzez połączenie tego przecięcia z przeciwległymi wierzchołkami kwadratu, bez dodawania punktu Steinera.



Siatka po siódmym kroku – na zielono zaznaczono odcinki dołożone przy triangulacji

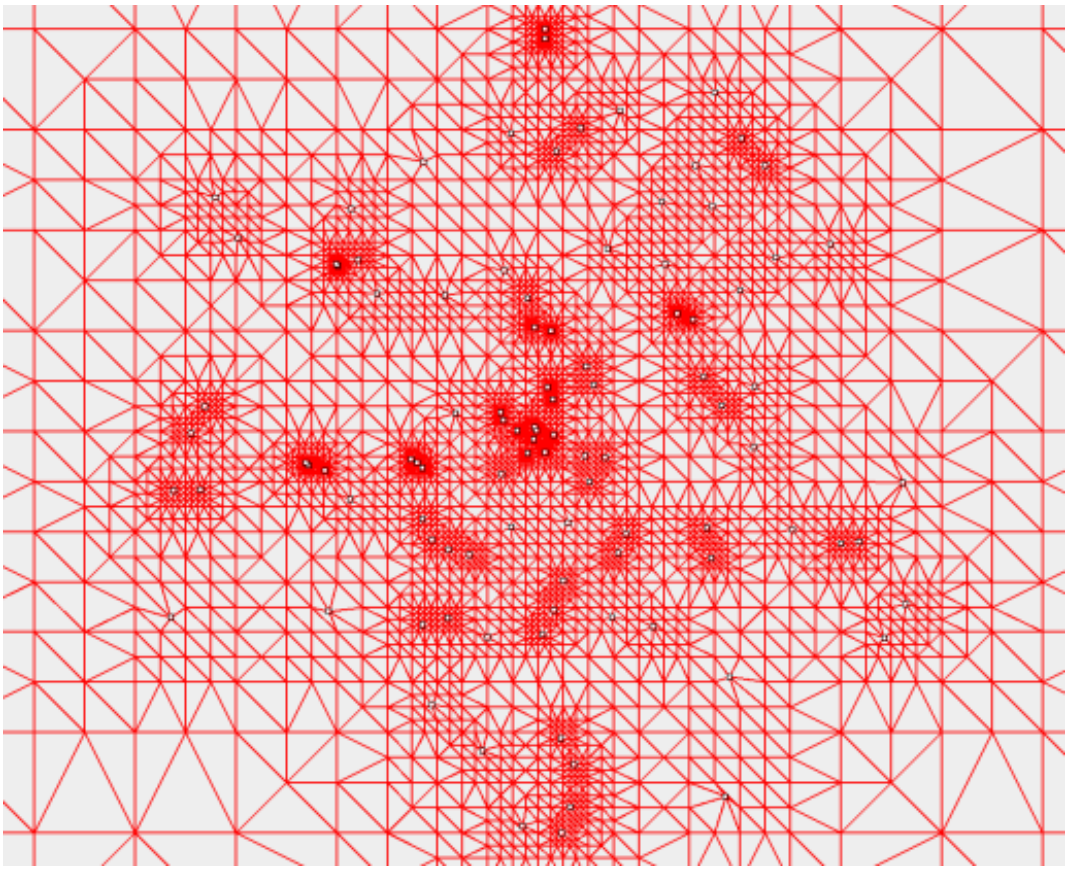
Wynik działania algorytmu

Poniżej zaprezentowano ostateczne wyjście algorytmu:

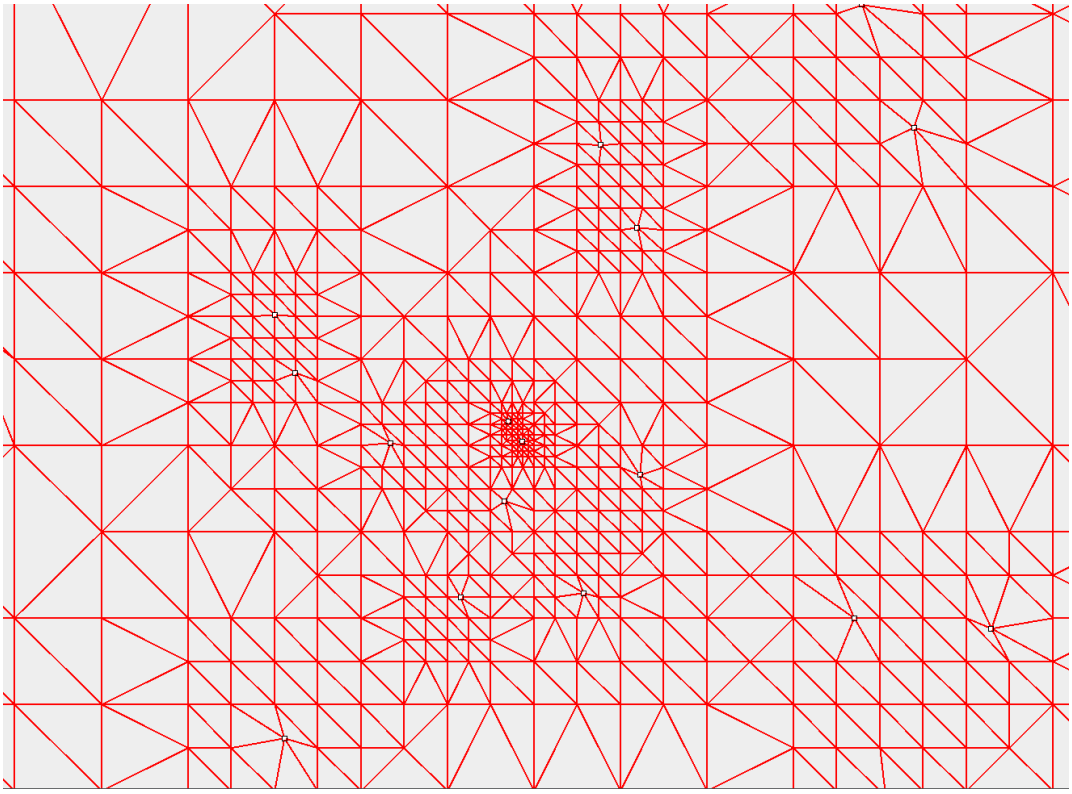


Ostateczna triangulacja – cała siatka

Widzimy, że dodanie punktów Steinera mocno zagęściło siatkę, liczbę trójkątów można znacznie zmniejszyć poprzez zastosowanie „kilku prostych heurystyk”[1], niestety nie zostały one opisane w artykule.



Ostateczna triangulacja – zbliżenie na punkty



Ostateczna triangulacja – zbliżenie na środek, na punkty leżące blisko siebie, widzimy, jak wzajemna odległość punktów wpływa na gęstość otrzymanej siatki

Bibliografia

[1]Bern, M., Eppstein, D., & Gilbert, J. (1994, Lipiec 8). Provably good mesh generation. *Journal of Computer and System Sciences* , pp. 384–409.

[2]de Berg, M., van Kreveld, M., Cheong, O., & Overmars, M. (2008). *Computational Geometry: Algorithms and Applications* (3 ed.). Berlin Heidelberg: Springer-Verlag.