УДК 51-74

Олексій ЧУНІХІН
alexey3322179@gmail.com
Тетяна ЧУНІХІНА
tchunihina1773t@gmail.com
м. Харків

ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПОБУДОВИ ПУАССОНІВСЬКОГО ДИСКУ МЕТОДОМ РОБЕРТА БРІДСОНА МОВОЮ С++

В статті описано алгоритм побудови Пуассонівського диску методом Роберта Брідсона у двовимірному просторі мовою С++. Оцінка алгоритму дорівнює O(N), так як для отримання N точок перегляд точок-кандидатів виконується рівно 2N-1 разів та не має вкладених циклів.

Встановлено, що описів методу Роберта Брідсона та його реалізації у україномовних джерелах немає. Алгоритм доцільно працює при виборі першої точки зі списку на кожній ітерації, що досягається генерацією k точок в околиці даної. Таким чином, можна замінити використання контейнеру з довільним доступом Vector на будь-який послідовний контейнер, наприклад, List.

***Ключові слова**: Пуассонівський диск, метод Роберта Брідсона, аналіз алгоритму, мова С++, контейнери STL.*

Постановка проблеми

Вибірка, яка створює точки, які щільно упаковані, але не ближче, ніж задана мінімальна відстань, що веде до більш природного рисунку точок, називається Пуассонівським диском. Та використовується у багатьох додатках графіки, зокрема у рендерингу, генерації зразків із синього розподілу шумів, ідеально підходить для багатьох застосувань у візуалізації.

Аналіз останніх досліджень і публікацій

Пуассонівський диск – це таке розподілення точок на площині, де всі точки знаходяться не менше, ніж на деякій фіксованій відстані одна від одної. Параметр щільності задається користувачем. Вибірка знаходить застосування у рендерингу, комп'ютерній графіці та візуалізації [1]. У цій статті розглядається реалізація диску Пуассона у двовимірному просторі для розподілення точок на площині «природним» чином. Для цього розглядається декілька алгоритмів і приводиться реалізація одного із них мовою С++.

Для побудови Пуассонівського диску існують алгоритм «найкращого кандидата» Мітчела [3] та алгоритм Даніеля Данбара та Грега Хамфріса [2] недоліком їх використання є або неможливість розширити алгоритм для довільного *N*-мірного простору [4], або низька швидкодія.

Метод побудови Роберта Брідсона було описано у статті «Fast Poisson Disk Sampling in Arbitrary Dimensions» [5], цей алгоритм дозволяє побудувати Пуассонівський диск у *N*-мірному просторі за лінійний час *O*(*N*). Описів даного алгоритму та його реалізації у україномовних джерелах немає. Завданнями даної роботи є: огляд роботи алгоритму, опис реалізації мовою С++, аналіз оптимальності використання STL контейнерів List та Vector для збереження множини потенційних точок та відстеження змін у побудові диску в залежності від способу вибору наступної точки для розгляду.

Постановка завдання

Мета статті є огляд роботи алгоритму, опис реалізації мовою С++, аналіз оптимальності використання STL контейнерів List та Vector для збереження множини потенційних точок та відстеження змін у побудові диску в залежності від способу вибору наступної точки для розгляду.

Виклад основного матеріалу

Нехай необхідно площину розмірами $n \times m$ замостити точками, що віддалені одна від одної не менше, ніж деяке minDist, в околі кожної точки будемо будувати k точок, де k – щільність побудови.

Створимо клас, що описує точку на двовимірній площині з координатами (x, y). Для даної точки доступні два види конструкторів та методи для перегляду поточних координат та метод inRectangle(), який перевіряє належність точки прямокутнику. Даний метод будемо використовувати для перевірки приналежності точки площині розмірами $n \times m$ (лістінг 1).

Опишемо особливості реалізації методу Роберта Брідсона мовою C++:

1. Поділимо нашу площину на сітку, розміром cellSize = minDist/√2.

При створенні нової точки, за допомогою цієї сітки, відсікаємо непотрібні перевірки відстаней до кожної вже існуючої точки і перевіряємо тільки ті точки, які знаходяться у сусідніх клітинках сітки (не більше 8 точок, тобто точки зліва/справа/зверху/знизу/по кутах сітки від заданої).

```
double cellSize = min_dist / sqrt(2);
```

2. Створимо двовимірний масив, який відповідає сітці, кожен елемент масиву – це точка, яка знаходиться в клітинці сітки з координатами (i, j). Оскільки масштаб клітинки більше мінімальної відстані, то гарантовано більше однієї точки не може знаходитися у клітинці сітки (лістінг 2).

3. Згенеруємо довільну точку і додамо її до контейнеру точок. Для цього використаємо контейнер Vector, що у подальшому дасть змогу звертатися до довільного елементу (лістінг 3).

4. Покладемо першу згенеровану точку у клітинку сітки, що відповідає координатам цієї точки. Розрахуємо координати клітинки (i, j), на основі залежності: i = x / cellSize, j = y / cellSize (лістінг 4).

Метод imageToGrid поверне пару чисел, що відповідатимуть координатам (i, j).

Звертатися до даних координат можна за допомогою полів first та second класу pair.

5. Доки контейнер точок не пустий:

```
while (!(processList.empty())) {…}
```

6. Дістаємо довільну точку із списку (лістінг 5).

Зауважимо, що вибір довільної точки не є обов'язковою умовою коректної роботи алгоритму. Тому маємо змогу обирати на кожній ітерації першу/останню точку, що дасть нам можливість замінити використання контейнера Vector на будь-який менш трудомісткий контейнер з послідовним доступом до елементів, наприклад, List або Queue.

```
Dot point = processList.top_front();
processList.pop_front();
```

7. k разів (де k – щільність, кількість точок у колі) генеруємо нову точку в околиці даної, яка знаходиться не ближче, ніж minDist і не далі, ніж maxDist (лістінг 6).

8. Для кожної згенерованої точки перевіряємо, що:

– координати не вийшли за границю площини $n \times m$:

```
newPoint.inRectangle(width, height);
```

– точка не лежить в околиці іншої точки. Тобто, перевіряємо тільки сусідні, загалом 8 точок (ліворуч-праворуч-зверху-знизу та по кутах) (лістінг 7).

9. Точку яка задовольняє дані умови, додаємо до контейнеру та відображаємо на екрані, а також кладемо у відповідну клітинку сітки (лістінг 8).

10. Повертаємося до пункту №5.

Оскільки на 6 кроці алгоритму необхідно мати швидкий доступ до довільної точки зі списку, то доцільно використовувати STL – контейнер Vector, тому що швидкість доступу до довільного елементу даного контейнера – O(const). Але у ході дослідження виявлено, що вибір довільної точки не є принциповим.

Протестуємо роботу алгоритму при виборі точок довільним чином, або послідовно за критеріями швидкодії та кількості побудованих точок Пуассонівського диску. Результати приведемо у табл. 1.

Тестування проведемо при розмірах площини: 500x500 пікс., 1000x1000 пікс., 10000x10000 пікс., 20000x20000 пікс.; мінімальна відстань між точками 10 пікс.

Лістінг 1

```
class Dot
{
private:
    pair<int, int> xy;
public:
    Dot() {
        srand(time(0));
        xy.first = (rand() % rand() % 101)/100.0;
        xy.second = (rand() % rand() % 101) / 100.0;
    }
    Dot(int x, int y) {
        xy.first = x;
        xy.second = y;
    }
    bool inRectangle(int width, int height) {
        if (xy.first >= width || xy.first <= 0 || xy.second >= height || xy.second <= 0)
            return false;
        return true;
    }
    int getX() { return xy.first; }
    int getY() { return xy.second; }
};
```

Лістінг 2

```
m = ceil(width / cellSize) + 2;
n = ceil(height / cellSize) + 2;
Dot **grid = new Dot*[n];
for (int i = 0; i < n; i++) {
    grid[i] = new Dot [m];
}
```

Лістінг 3

```
vector<Dot> processList;
srand(time(0));
int x = firstPoint.getX();
int y = firstPoint.getY();
processList.push_back(firstPoint);
```

Лістінг 4

```
pair<int, int> imageToGrid(Dot point, double cellSize)
{
    int gridX = (int)(point.getX() / cellSize);
    int gridY = (int)(point.getY() / cellSize);
    return pair<int, int>(gridX, gridY);
}
pair<int, int> tmp = imageToGrid(firstPoint, cellSize);
grid[tmp.first][tmp.second]=(firstPoint);
```

Лістінг 5

```
int k = processList.size();
int randN = rand() % k;
Dot point = processList[randN];
processList.erase(processList.begin() + randN);
```

Лістінг 6

```
for (int i = 0; i < new_points_count; i++)
    Dot newPoint = generateRandomPointAround(point, min_dist);
```

Лістінг 7

```
!inNeighbourhood(grid, newPoint, min_dist, cellSize, sizePoint);
bool inNeighbourhood(Dot **grid, Dot newPoint, int min_dist,
                     double cellSize, int sizePoint) {
    pair<int, int> tmp = imageToGrid(newPoint, cellSize);
    int ti = tmp.first;
    int tj = tmp.second;
    if (ti >= mm || tj >= nn)return 0;
    int x = grid[ti][tj].getX();
    int y = grid[ti][tj].getY();
    if (distance(newPoint.getX(), newPoint.getY(), x, y, sizePoint) < min_dist)
        return true;
    /* Приведемо приклад розгляду правої точки від заданої. Останні точки розглядаються
       аналогічно: права точка (ti + 1), ліва (ti-1), верхня (tj-1) і т.д. */
    if(ti + 1<=mm) {
        int x = grid[ti + 1][tj].getX();
        int y = grid[ti + 1][tj].getY();
        if (distance(newPoint.getX(), newPoint.getY(), x, y, sizePoint) < min_dist)
            return true;
    }
}
```

Лістінг 8

```
processList.push_back(newPoint);
printDot(newPoint, sizePoint);
pair<int, int> tmp = imageToGrid(newPoint, cellSize);
grid[tmp.first][tmp.second]=(newPoint);
```

Табл. 1. Відстеження змін у побудові диску в залежності від способу вибору наступної точки для розгляду.

Розміри площини, пікс	Параметри	Довільна точка	Послідовна точка
500x500	Кількість побудованих точок, шт	1721	1727
	Швидкодія, мс	3619	3319
1000x1000	Кількість побудованих точок, шт	6757	6809
	Швидкодія, мс	14443	13278
10000x10000	Кількість побудованих точок, шт	673231	678877
	Швидкодія, мс	57756	31735
20000x20000	Кількість побудованих точок, шт	2692142	2712865
	Швидкодія, мс	337514	128352

Зауважимо, що кількість побудованих точок не залежить від способу вибору потенційних точок (рис. 1).

Алгоритм доцільно працює при виборі першої точки зі списку на кожній ітерації що досягається генерацією k точок в околі даної. Таким чином, можна замінити контейнер з довільним доступом Vector на будь-який послідовний контейнер, наприклад, List (рис. 2).

ISSN 2520-2820 (online)
ISSN 2524-0978 (print)

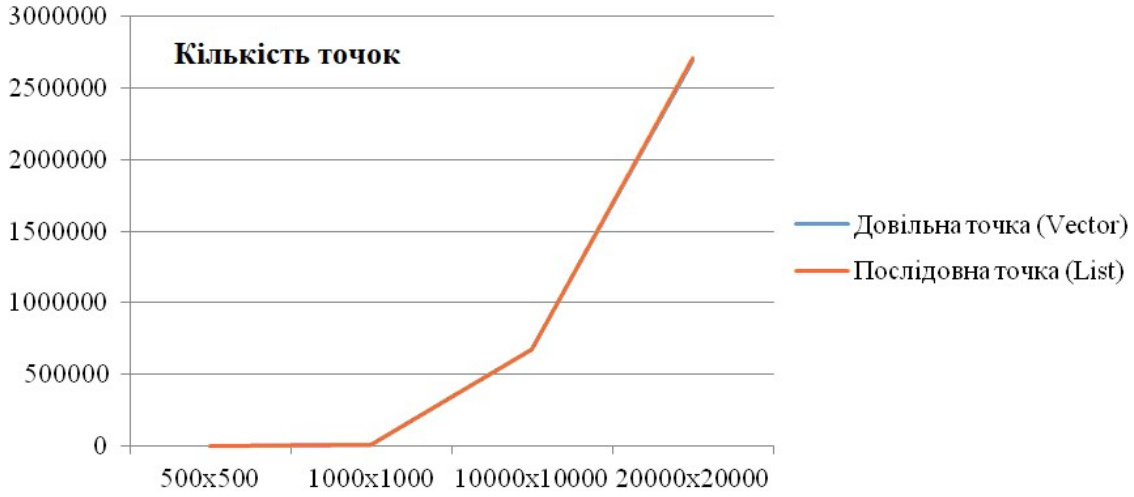

Рис. 1. Аналіз роботи алгоритму при зміні способу вибору точки за критерієм кількість побудованих точок

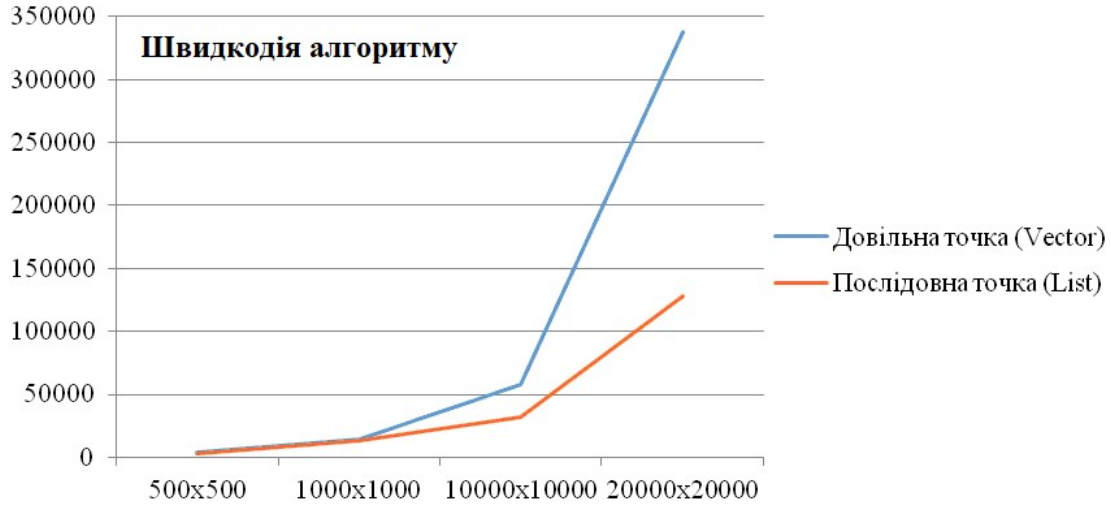

Рис. 2. Аналіз роботи алгоритму при зміні способу вибору точки за критерієм швидкості

Зауважимо, що для отримання N точок, крок 5 виконується рівно $2N - 1$ разів. Причому на кожній ітерації або отримується нова точка і додається до списку або видаляється існуюча точка зі списку. В кожній ітерації відсутні вкладені цикли, окрім кроку 6, який виконується константу кількість разів, чим можна знехтувати. Таким чином, оцінка алгоритму дорівнює $O(N)$.

Висновки і перспективи досліджень

Отже, ми розглянули алгоритми побудови Пуассонівського диску та описали метод Роберта Брідсона у двовимірному просторі. Досліджено зміни у побудові диску в залежності від способу вибору наступної точки для розгляду, при цьому відстежена різниця в швидкодії роботи алгоритму та кількості згенерованих точок. Можна зробити висновок, що для реалізації мовою C++ достатньо використовувати контейнер послідовного доступу List бібліотеки STL, так як у процесі аналізу виявлено, що при виборі наступної точки для розгляду випадковим чином (в контейнері Vector) та послідовної на кожній ітерації суттєвих відмінностей в згенерованих Пуассонівських дисках не виявлено. Тобто

кількість згенерованих точок приблизно однакова. При цьому, швидкість роботи алгоритму при використанні контейнеру List суттєво зростає в залежності від збільшення об'єму даних. Для подальшого дослідження планується розглянути роботу даного методу у тривимірному просторі і проаналізувати заявлену масштабованість алгоритму.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cook, R. L. Stochastic sampling in computer graphics [Text] / R.L. Cook // ACM Transactions on Graphics. – Vol. 5, No. 1. – 1986. – P. 51-72.
2. Dunbar, D. A Spatial Data Structure for Fast Poisson-disk Sample Generation [Text] / D. Dunbar, G. Humphreys // ACM Transactions on Graphics. – Vol. 25, No 3. – 2006. – P. 503-508.
3. Mitchell's Best-Candidate [Електронний ресурс]. – Режим доступу до ресурсу: https://bl.ocks.org/mbostock/1893974 – Дата звертання 20.09.2018.
4. Random Points on a Sphere [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jasondavies.com/maps/random-points/ – Дата звертання 19.09.2018.
5. Robert Bridson, Fast Poisson Disk Sampling in Arbitrary Dimensions University of British Columbia [Електронний ресурс]. – Режим доступу до ресурсу: https://www.cs.ubc.ca/~rbridson /docs/bridson-siggraph07-poissondisk.pdf – Дата звертання 20.09.2018.

Olexsii CHUNIKHIN, Tetiana CHUNIKHINA
Kharkiv

PECULIAR FEATURES OF IMPLEMENTATION FOR GENERATION OF THE POISSON DISK IN TERMS OF LANGUAGE C++

The paper describes the algorithm for constructing of a Poisson disk using the Robert Bridson method in two-dimensional space with C++ language. The estimation of the algorithm is equal to O(N), since for obtaining N points review of the candidate points is carried out exactly 2N-1 times and does not have embedded cycles.

It is established that there are no descriptions of the Robert Bridson method and its implementation in Ukrainian-language sources. The algorithm works expediently when selecting the first point in the list for iteration which is achieved by generating "k" points in neighboring of this one. This way, you can replace usage of the container with a free access Vector to any successive container, such as e. g. the List.

Keywords: *Poisson disk, the Robert Bridson method, analysis of the algorithm, C++ language, STL containers.*

Алексей ЧУНИХИН, Татьяна ЧУНИХИНА
Харьков

ОСОБЕННОСТИ РЕАЛИЗАЦИИ ПОСТРОЕНИЯ ПУАССОНОВСКОГО ДИСКА МЕТОДОМ РОБЕРТА БРИДСОНА НА ЯЗЫКЕ С++

В статье описан алгоритм построения Пуассоновского диска методом Роберта Бридсона в двумерном пространстве языком С++. Оценка алгоритма равна O(N), так как для получения N точек пересмотр точек-кандидатов исполняется ровно 2N-1 раз и не имеет вложенных циклов.

Установлено, что описаний метода Роберта Бридсона и его реализации в укриноязычных источниках нет. Алгоритм уместно работает при выборе первой точки из списка на каждой итерации, что достигается генерацией k точек в окрестности данной. Таким образом, можно заменить использование контейнера с произвольным доступом Vector на любой последовательный контейнер, например, List.

Ключевые слова*: Пуассоновский диск, метод Роберта Бридсона, анализ алгоритма, язык С++, контейнеры STL.*

Стаття надійшла до редколегії 26.09.2018