Poisson-Disc Sampling Set

Matteo Zacchino 1913421

31 gennaio 2023

Abstract - Poisson Point Set è una tecnica molto utilizzata che permette di distribuire in maniera uniforme dei "sample" in dimensioni arbitrarie. All'interno del progetto si è utilizzato uno spazio bidimensionale.

---

1 Introduzione

L'efficienza del Poisson Disc Sampling Set risiede nel fatto che, a differenza del Dart Throwing dove ogni volta che un nuovo "sample" viene generato si controlla se è abbastanza distante tra tutti gli altri "sample", esso utilizza una griglia n-dimensionale in cui ogni cella ha una grandezza stabilita attraverso il teorema di Pitagora (Figura 1).

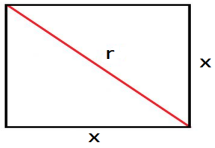


Figura 1: Calcolo della diagonale

Così facendo ogni cella della griglia potrà contenere al massimo un solo "sample" per volta. Quando si genererà un nuovo "sample" sarà sufficiente controllare la distanza solamente con i "sample" delle celle adiacenti e non più con tutti gli altri "sample". Ottenendo un algoritmo dell'ordine di $O(n)$ e non $O(n^2)$ come il Dart Throwing. Ulteriori dettagli si trovano nel paper scritto da Bridson [1].

2 Implementazione

L'algoritmo è stato implementato in Python e segue passo dopo passo l'algoritmo descritto nel paper [1] generando un file .svg raffigurante la generazione dei "sample". Per lo sviluppo dell'algoritmo si è scelta la libreria *svgwrite* che permette di disegnare forme, in questo caso i "sample" sono mostrati come punti e di salvare tutto ciò che si è disegnto in un file .svg. Un'alternativa poteva essere quella di utilizzare la libreria *matplotlib* e creare dei plot dei "sample" come punti.

2.1 Spiegazione Algoritmo

L'algoritmo sostanzialmente segue i 3 step descritti in [1]:

1. In base ai parametri r, k, *width* e *height* che sono rispettivamente la distanza minima dei punti, il numero massimo di generazioni di nuovi "sample", la larghezza dello schermo e l'altezza dello schermo, si calcola la cellsize utilizzando la formula della Figura 1, da questa si hanno il numero di righe e di colonne della griglia principale che viene inizializzata con tutti -1.
2. Si generano uniformemente al dominio (R^2) 2 punti x e y che vengono salvati in un vettore a 2 dimensioni, il quale viene memorizzato nella griglia principale come nuovo "sample" e inserito nella lista "attiva" come primo "sample" da cui partire.

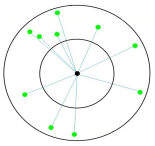
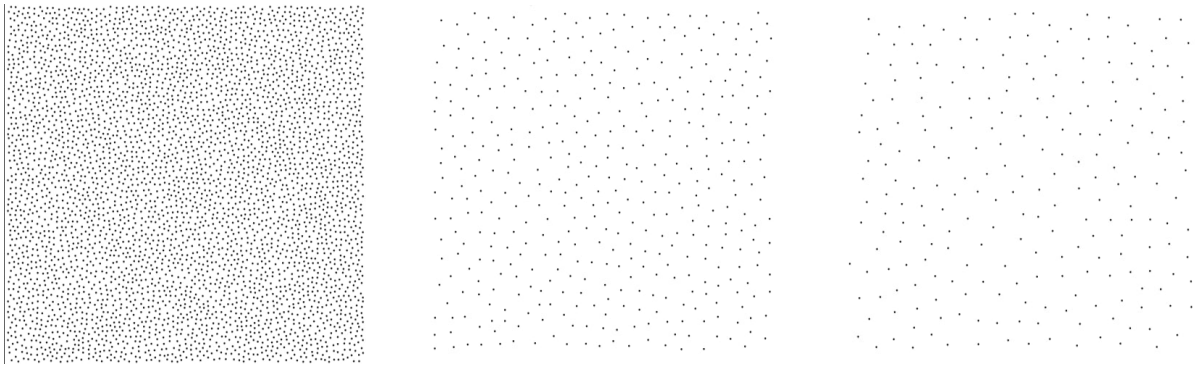


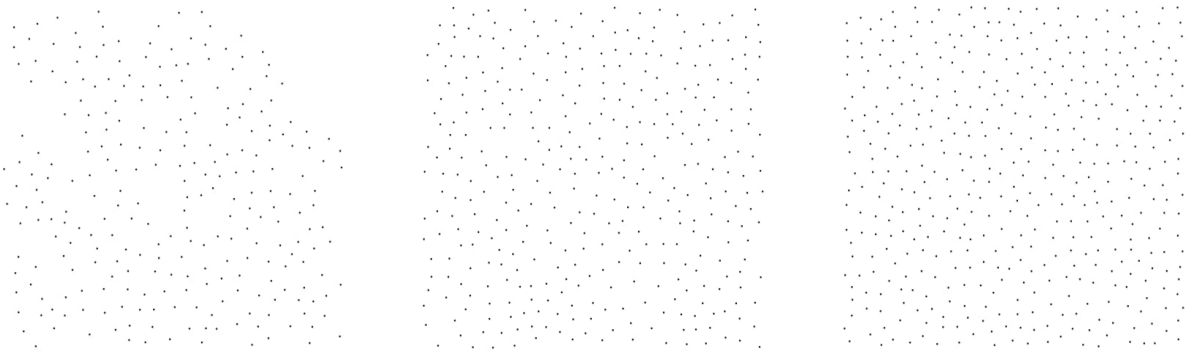
Figura 2: Generazione di vettori entro la corona sferica di dimensioni r-2r

3. Finchè si hanno ancora "sample" nella lista "attiva", si generano fino a k nuovi "sample", che siano entro il raggio della corona circolare che va da r a 2r. Per farlo si generano vettori con angolazione casuale, si normalizzano e si imposta la loro lunghezza con un valore che rientri nel range appena descritto e si cambia la loro origine nel "sample" selezionato (Figura 2). Dopodiché si controlla se questi nuovi "sample" siano abbastanza distanti dai loro vicini, così da diventare nuovi "sample" ed essere inseriti nella griglia principale. Per



(a) $r = 5, k = 30$ (b) $r = 15, k = 30$ (c) $r = 20, k = 30$

Figura 3: Esempi di risultati con distanza r variabile



(a) $r = 15, k = 2$ (b) $r = 15, k = 5$ (c) $r = 15, k = 50$

Figura 4: Esempi di risultati con tentativi k variabili

farlo si analizzano tutte le celle della griglia principale adiacenti al "sample" selezionato che contengono altri "sample" e si controlla la loro distanza.

4. Infine si generano sullo schermo puntini neri, per raffigurare i "sample" nella griglia principale. I puntini hanno le coordinate dei "sample" e la grandezza di 1 pixel. E si salva il risultato all'interno di un file .svg.

3 Risultati

In alto sono mostrati alcuni dei risultati prodotti dall'algoritmo generando immagini in cui varia la distanza minima tra i punti e il numero di tentativi effettuati all'interno di finestre grandi 400x400. Nella Figura 3 sono mostrati 3 esempi in cui si è eseguito l'algoritmo variando ogni volta il valore r, si nota semplicemente che riducendo la distanza tra i punti, minore sarà la distanza, maggiore sarà il numero di punti generati. La cosa importante è che i punti risultano essere distribuiti uniformemente garantendo sempre la distanza minima indicata e non avendo mai sovrapposizioni di punti. Nella Figura 4, invece, sono mostrati tre esempi in cui si è eseguito l'algoritmo variando ogni volta il valore k. In questi si vede bene che riducendo i tentativi di generazione di punti attorno ad ogni "sample", portandoli ad un valore inferiore di 30, si può incappare in spazi molto meno densi di altri, anche vuoti. Aumentando invece a più di 30 il numero di tentativi per la generazione dei punti la differenza diventa sempre meno evidente, quasi impercettibile.

4 Considerazioni finali

É stato divertente e interessante realizzare questo progetto. Ho trovato il paper di Robert Bridson [1] molto esplicativo, inoltre è stato molto d'aiuto anche il video pubblicato su YouTube da *The Coding Train [2]*, grazie al quale ho compreso meglio l'algoritmo, dalla *documentazione [3]* sono riuscito ad utilizzare la libreria svgwrite e dal *Video Youtube [4]* ho preso spunto solo ed esclusivamente per la classe *Vector2*. Ritengo che i risultati finali siano abbastanza ragionevoli e che l'algoritmo funzioni adeguatamente. Tramite quest'ultimo, utilizzando la libreria *bpy [5]*, sono riuscito a creare una scena

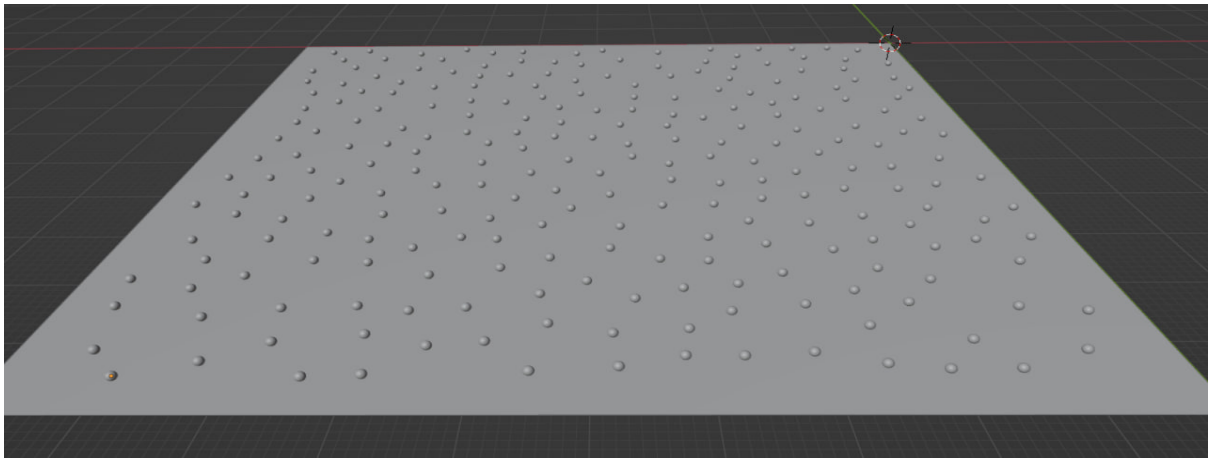


Figura 5: Immagine renderizzata della generazione di sfere su un piano

Figura 6: Immagine renderizzata della generazione di una Foresta

3D renderizzabile in blender che inserisce i "sample" generati dall'algoritmo, sotto forma di sfere, su un piano, Figura 5. Successivamente prendendo come riferimento il *video di Youtube [6]* in cui ho capito come generare oggetti, la libreria *bpy [5]* per eseguire l'append di altri modelli blender nel file e il *sito Free3D [7]* che ho utilizzato per scaricare il modello di un albero, sono riuscito a renderizzare tramite l'algoritmo leggermente modificato una foresta, Figura 6. L'obiettivo finale sembra essermi riuscito, anche se per creare il rendering l'algoritmo non impiega poco tempo. Questi ultimi due script python eseguono l'algoritmo di *Bridson [1]* contenuto nel file python *PoissonDiscSampling.py* e utilizzando la griglia che contiene le posizioni di tutti i "samples", generano gli oggetti desiderati all'interno della scena. Alla fine della generazione salvano nella stessa cartella un file .obj contenente la scena realizzata. Per eseguire gli script si deve aprire il file .blender che si desiderano e andare nella sezione "Scripting" in alto a destra. Aprendo il file lo script dovrebbe essere già presente, in caso contrario aprire *Sphere_Script.py* per la generazione delle sfere sul piano, oppure *Forest_Script.py* per la generazione della foresta.

References

[1] R Bridson - SIGGRAPH sketches, 2007, “Fast Poisson Disk Sampling in Arbitrary Dimensions” https://www.cs.ubc.ca/ rbridson/docs/bridson-siggraph07-poissondisk.pdf
[2] The Coding Train, Coding Challenge #33- Poisson-disc Sampling, 2018, Video Youtube
[3] Svgwrite 1.4.3 documentation Documentazione
[4] Auctux, Poisson Disk Sampling (c++ e python) Video Youtube
[5] Blender 3.4 Python API Documentation, Documentazione
[6] Prospero Blender, How to Spawn Objects at Specific Locations with Python in Blender, Video Youtube
[7] Free3D, Sito Internet