Poisson Point Set

Riccardo Pitzanti
Sapienza Università di Roma

Abstract

Una breve panoramica sul "sampling", volta a descrivere un possibile algoritmo per la tecnica "Poisson Point Set", con annessi esempi e codice Python.

1 Introduzione

Il campionamento, o "sampling" è una procedura spesso adottata in informatica. Il campionamento traduce un segnale continuo nella sua controparte discreta o seleziona un sottoinsieme da un insieme discreto di segnali, in modo che il segnale possa essere rappresentato ed elaborato dai computer in modo efficiente. Per esempio, le onde sonore unidimensionali, le immagini bidimensionali e le polygonal meshes tridimensionali vengono acquisite mediante campionamento discreto da segnali continui. Nella computer grafica, il campionamento svolge un ruolo importante in molte applicazioni, come il rendering, lo stippling, la sintesi di texture, la distribuzione di oggetti e la simulazione. È possibile classificare le distribuzioni di punti osservando lo spettro di Fourier. Spettri diversi sono associati a colori diversi. Ad esempio, alcune classificazioni possibili sono: "white-noise", "blue noise", "pink-noise" e "brown-noise". Tra tutte le tecniche di campionamento, il campionamento "blue-noise" è quella più diffusa, dato che genera distribuzioni uniformi randomizzate. Il campionamento a disco di Poisson è una tecnica che genera insiemi di punti/dischi distribuiti in modo uniforme e casuale. Un insieme ideale di punti campionati a disco di Poisson nel dominio di campionamento Ω deve soddisfare le seguenti tre proprietà: 1) la proprietà della distanza minima, che richiede che la distanza tra i centri di due dischi qualsiasi sia più grande del raggio di campionamento; 2) proprietà di campionamento imparziale, che richiede che ogni sezione del dominio abbia una probabilità proporzionale al dimensionamento di questa sezione di ricevere un punto di campionamento; 3) proprietà di campionamento massimo, che richiede che l'unione dei dischi copra l'intero dominio di campionamento. Il campionamento è uniforme se il raggio di campionamento è costante, in caso contrario è un campionamento adattivo. Il metodo tradizionale per il campionamento a disco di Poisson è chiamato "dart-throwing" ed è stato proposto per la prima volta da Cook. Dato un dominio di campionamento e un raggio di campionamento, l'algoritmo genera dischi nel dominio di campionamento in modo casuale. Se il disco correntemente generato è in conflitto con uno dei dischi campionati in precedenza, viene rifiutato; altrimenti, viene accettato. Questo procedimento viene ripetuto finché non si raggiunge un numero di rifiuti fissato a priori. La complessità dell'algoritmo "dart-throwing" originale è $O(n^2)$; tuttavia, questo approccio è inefficiente. Robert Bridson (University of British Columbia) ha esposto un nuovo algoritmo avente complessità di tempo $O(N)$, dove N indica il numero di punti che vengono generati. L'algoritmo, focus di questo progetto, è applicabile indipendentemente dal numero di dimensioni del piano su cui si vogliono distribuire i punti.

2 Algoritmo

L'algoritmo richiede i seguenti parametri: il numero di dimensioni del piano su cui verranno posizionati i punti (n), la distanza minima tra i punti (r), il numero di tentativi di posizionamento di punti intorno al punto di riferimento attuale (k). Come parte del processo di inizializzazione, viene creata una griglia utile a memorizzare i punti nello spazio e a velocizzare le ricerche intorno a ciascun punto. Il lato di ogni cella della griglia è di lunghezza $\frac{r}{\sqrt{n}}$, questo perché ogni cella dovrà contenere al massimo un punto. La griglia viene implementata come un array n-dimensionale di interi: il valore -1 indica la mancanza di un punto in quella cella, un intero non negativo indica l'indice di un punto collocato in una cella. Inoltre, viene selezionato e posizionato in una cella, scelta in modo casuale e uniforme dal dominio, un punto da cui partire, che verrà utilizzato come punto iniziale della lista dei punti attivi (lista di indici dei punti, dunque il primo sarà 0). Nella fase successiva dell'algoritmo, è previsto un ciclo: inizialmente, data la lista di punti attivi di lunghezza l, viene scelto a caso un indice compreso tra 0 e $l-1$. Successivamente, vengono effettuati k tentativi per posizionare nuovi punti intorno al punto di partenza. Quando si tenta di generare un nuovo punto, si sceglie a caso una cella posizionata nell'area delimitata dalla differenza delle aree di due circonferenze, ovvero quelle aventi raggio, rispettivamente, $2r$ e r, con centro nel punto sorgente. In questo modo i nuovi punti saranno sempre distanti almeno r dal punto scelto. Per campionare un punto all'interno di quest'area, vengono sfruttate le coordinate polari, rappresentate da, nel caso a due dimensioni, una distanza e un angolo rispetto al punto di riferimento. Utilizzando il punto selezionato come punto di riferimento, viene scelto randomicamente un angolo e un raggio compresi tra r e $2r$, per poi attuare la conversione in coordinate cartesiane. Generato il nuovo punto, l'algoritmo analizza la cella che lo dovrebbe ospitare e si assicura che il punto da inserire sia al di fuori del raggio dei vicini relativi al punto generatore, senza dover controllare tutti i punti; verifica inoltre che vengano rispettati i limiti di larghezza/altezza del piano. Se non vengono trovati vicini o se il punto è al di fuori del raggio di ciascun vicino, il punto è valido, dunque viene disegnato e aggiunto alla lista dei punti attivi. Infine, tutti i punti sorgente che non sono stati in grado di generare punti validi sono rimossi dalla lista dei punti attivi, fino a quando quest'ultima non sarà vuota.

3 Analisi dei risultati

Di seguito, alcuni esempi generati da un'implementazione in Python dell'algoritmo descritto. Sono state realizzate due versioni di questa implementazione, una per piani 2D e una per piani 3D. Il codice è dunque facilmente generalizzabile, per far sì che si possano generare risultati in più dimensioni. I punti sul piano sono stati colorati, in modo tale da distinguere i punti da cui si è partiti (colori caldi) e i punti generati via via con l'avanzamento dell'algoritmo (colori freddi).

3.1 Analisi esempi bidimensionali

Negli esempi bidimensionali generati, si può notare il corretto funzionamento dell'algoritmo, dapprima testato con k = 30 e r = 1.7 (a). Aumentando il raggio r, i punti risultano più distanti fra loro come previsto (b) anche se, aumentando altezza e lunghezza del piano (100, 100 anzichè 60, 45) e lasciando invariati i parametri, i punti sembrano più vicini. Diminuendo la distanza, con r = 0.5, i punti formano una nube di colore, conseguenza della lontananza ridotta.

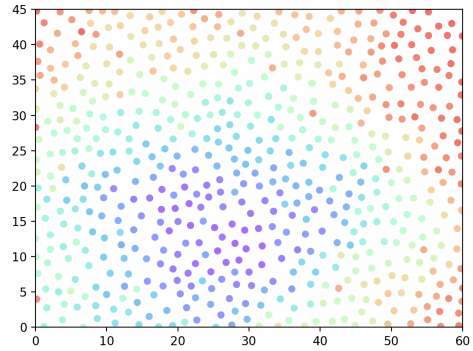
3.2 Analisi esempi tridimensionali

Spostando l'attenzione sugli esempi in 3D a pagina 3, l'algoritmo è stato inizialmente eseguito con parametri k = 30 e r = 7.5 (e). Aumentando i valori delle tre dimensioni (dunque passando da 60 a 100 per ciascuna delle tre), vengono disegnati più punti rispetto all'esempio precedente (g). Decrementando il valore di r (precisamente variato a 4.4), è possibile notare che i punti appaiono molto meno distanti fra loro (f), mentre se si aumenta il valore di k (ora impostato a 90), il risultato non sembra variare (g).

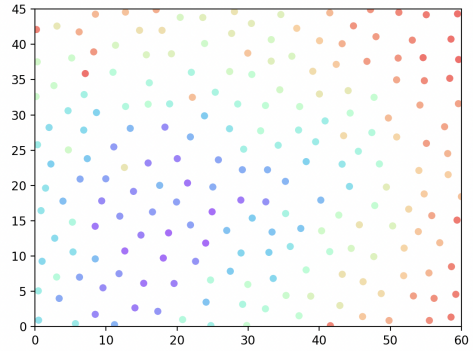
4 Riferimenti e risorse utilizzate

A questo documento sono state allegate le immagini d'esempio generate dalle implementazioni Python menzionate in precedenza, insieme a due file .py in cui è presente il codice stesso delle implementazioni. Di seguito, un elenco di risorse web consultate durante la realizzazione di questo progetto:

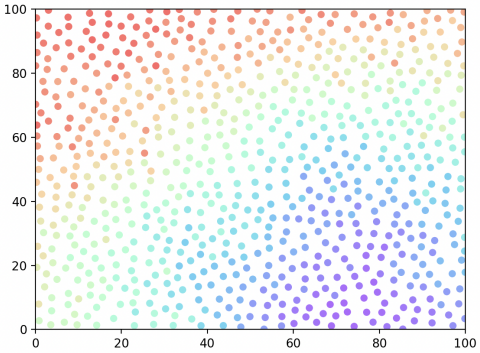
https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf
https://www.youtube.com/watch?v=flQgnCUxHlw&t=349s&ab_channel=TheCodingTrain
https://jianweiguo.net/publications/papers/2015_JCST_BNMeshSurvey_compress.pdf
https://scipython.com/blog/poisson-disc-sampling-in-python/
https://a5huynh.github.io/posts/2019/poisson-disk-sampling/



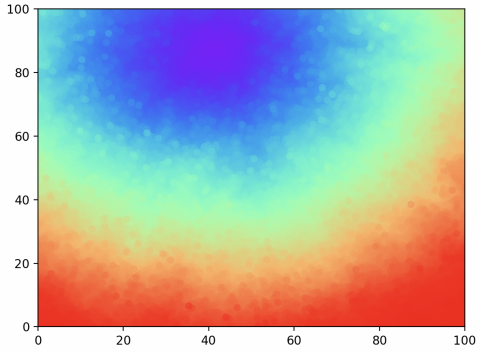
(a)



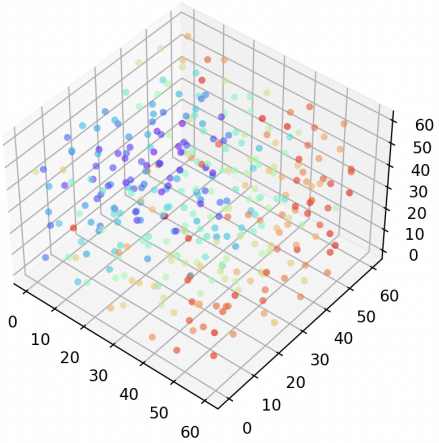
(b)



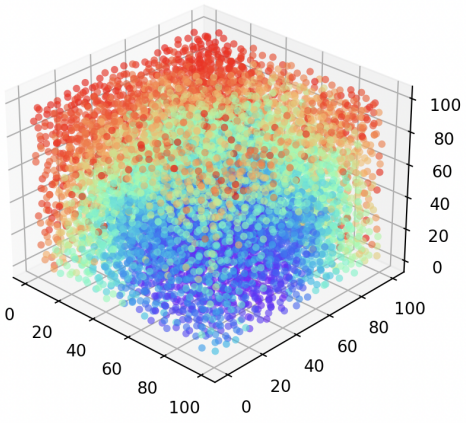
(c)



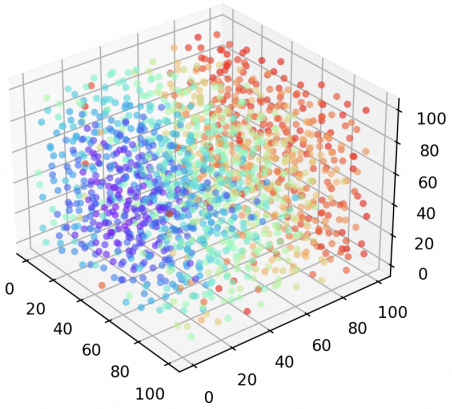
(d)


0 10 20 30 40 50 60
0 10 20 30 40 50 60
0 10 20 30 40 50 60

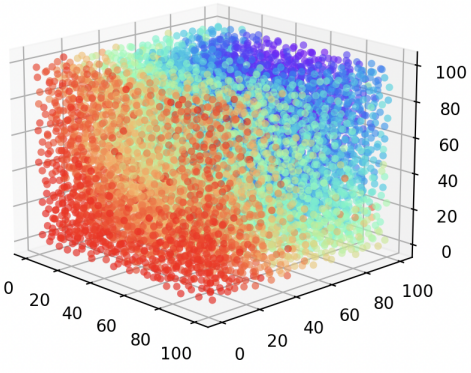
(e)


0 20 40 60 80 100
0 20 40 60 80 100
0 20 40 60 80 100

(g)


0 20 40 60 80 100
0 20 40 60 80 100
0 20 40 60 80 100

(f)


0 20 40 60 80 100
0 20 40 60 80 100
0 20 40 60 80 100

(h)