2023

Metoder för procedurell generering av 3D miljöer

OLIVER SVENSKBERG

HANDLEDARE: MATS ASPNÄS

INNEHÅLLSFÖRTECKNING

1. Inledning

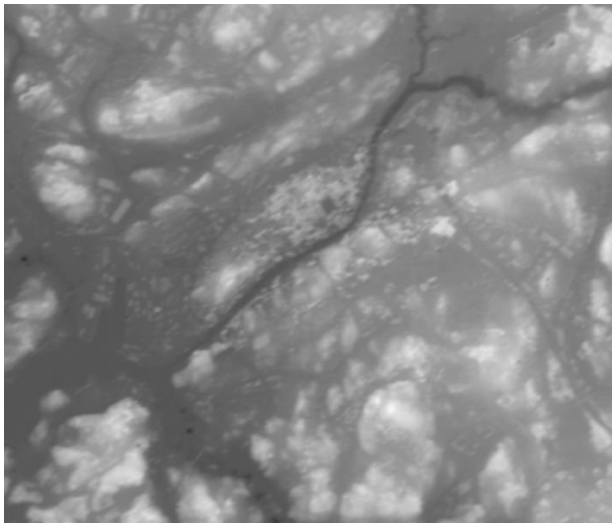
Många moderna spel och filmer utspelar sig i stora och detaljerade världar. I flera fall finns det inte färdiga digitala omgivningar som tillverkare kan använda sig av. Den moderna standarden för dessa omgivningar är hög för produkter med stora budgetar. Det betyder att det finns ett stort behov av kostnadseffektiva metoder för att skapa dessa omgivningar. För att möta den moderna standarden, måste omgivningarna reflektera det man ser i verkligheten. Ett berg kan vara snöigt och ha flera höga klippor, medan en skog kan bestå av flera trädarter, en rik skogsbotten, en rinnande bäck osv. I flera fall kan det vara omöjligt att skapa sådana omgivningar för hand, eftersom de helt enkelt är för stora och därför kräver mycket tid [1].

Det mest grundläggande egenskapen man måste skapa då man designar en digital omgivning är terrängen. En digital värld kan vara verklighetstrogen på alla sätt, men om inte terrängen är tillräckligt varierad, detaljerad eller väl lämpad för miljön, ser man det oftast ganska snabbt. För att lösa problemet med att skapa stora och detaljerade omgivningar på ett kostnadseffektivt sätt, har man skapat procedurell innehållsgenerering (*PCG, eng. Procedural Content Generation*) [2]. Syftet med denna avhandling är att presentera de mest grundläggande metoderna för att generera terräng och övriga metoder som stöder dem.

2. Procedurell innehållsgenerering

Inom datavetenskap syftar ordet *procedurell* till egenskaper hos entiteter som beskrivs eller skapas av datorkod [4]. PCG innebär att generera innehåll med hjälp av parametriserade algoritmer som har justerats för sina ändamål. Praktiskt taget kan man generera nästan vad som helst. I spel är det vanligt att terräng, karaktärer, nivåer och regler genereras slumpmässigt med hjälp av PCG [1]. Det finns flera fördelar med att använda sig av PCG i stället för att manuellt skapa innehåll. I spel kan man till exempel generera nivåer. Det kan innebära att spelaren möter nya layouter varje gång som hen spelar, vilket i tur gör spelet mer omväxlande. På detta sätt kan man också minska minnesmängden som spelet tar upp på datorminnet, eftersom nivåerna är inte sparade individuellt i sin helhet.

2D texturer kan också genereras för olika ändamål. Digitala konstnärer kan spara mycket tid på att använda till exempel brusfunktioner för att generera texturer.



Figur 1, en höjdkarta av Åbo, Finland. Skapad med Heightmapper-verktyget på https://tangrams.github.io/heightmapper/

Man kan även generera texturer vilket inte upprepar sig, som naturligtvis får omgivningen att se mer verklighetstrogen ut.

Texturer kan också användas för att skapa 3D-terräng. Höjdkartor är bilder eller texturer som representerar höjdvärden för terräng på ett tvådimensionellt sätt [4]. Varje cell i höjdkartan innehåller ett höjdvärde. En vanlig metod för det här är att färga varje cell. Ljusare färgtoner representerar oftast höga positioner, medan mörkare är lägre, se figur 1. Med höjdkartor kan man lagra höjdvärden för terräng på ett kompakt sätt, men största nackdelen är att en cell endast kan ha ett höjdvärde. Det betyder att saker som har överhäng, till exempel grottor, inte kan genereras i terrängen med höjdkartor. Ett sätt att lösa det här är att spara cellvärden i tre dimensioner i stället för bara två. Voxlar liknar pixlar, men de representerar värden i tre dimensioner [1]. I samband med terränggerenering, behöver en voxel endast innehålla information om ifall den positionen är terräng eller inte. Utöver det kan man lagra annan information i voxeln, till exempel om den är en del av en byggnad eller ett annat föremål i världen. Nackdelen med voxlar är att de kräver mera datorminne, eftersom de sparar terrängdata i flera dimensioner än pixlar.

3. Brus

Brus är ostrukturerad information som finns i flera former, men inom datorgrafik syftar man oftast till procedurella texturer. De är icke-reguljära primitiva funktioner, som skapar ostrukturerade mönster. Dessa funktioner passar bra till att generera naturliga omgivningar, eftersom upprepade mönster sällan förekommer i naturen [6].

Procedurella brusfunktioner är metoder som har anpassat en eller flera vanliga brusfunktioner till att generera information. De kan generera brus i alla resolutioner utan några synliga gränser. Funktionerna är parametriserade så att man kan ändra på stilen av bruset. Algoritmerna körs i konstant tid, eftersom varje datapunkt genereras enligt samma villkor. De finns många potentiella fördelar med procedurella brusfunktioner, men det är inte alltid nödvändigt att implementera all funktionalitet [4].

Det är viktigt att veta att äkta slumpmässighet är relativt sällsynt inom datavetenskap, och de brus som behandlas i denna text är pseudo-slumpmässiga. De genereras på ett sätt som imiterar äkta slumpmässighet, men i verkligheten är de inte det. Dessa algoritmer är väldigt invecklade för att resultaten ska bli så oförutsägbara som möjligt. I praktiken är de tillräckligt kaotiska för alla tänkbara ändamål [6, p.67].

I detta kapitel beskrivs olika sorters brus som används för PCG. Det är värt att notera att alla brustyper som diskuteras i texten går att implementera i minst tre dimensioner. I texten beskrivs de i bara två dimensioner för att det är lättare att förstå.

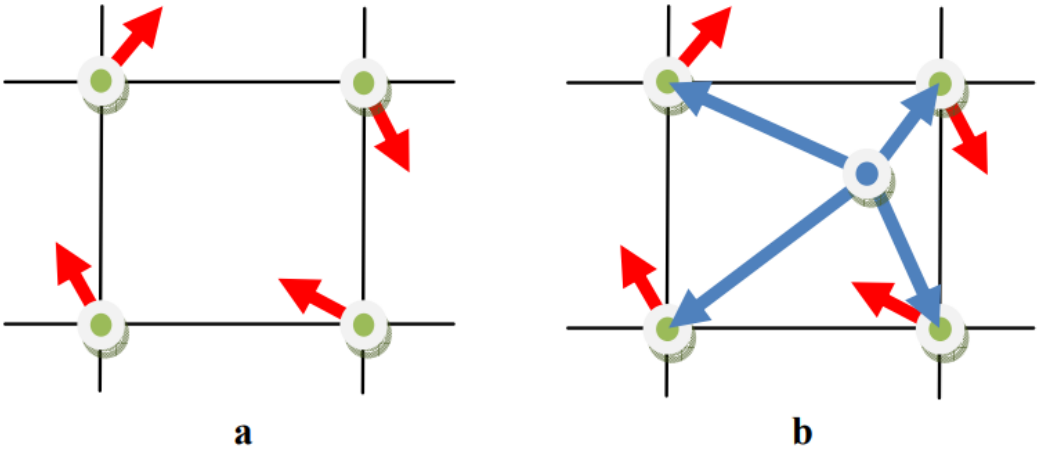
3.1 Vitt brus

Det enklaste och mest allmänna bruset är vitt brus. Tv-skärmar brukade visa vitt brus då de var inställda på en kanal som inte upptog en sändning. Ljudsignalen som tv:n fick då var en jämn distribution av alla frekvenser, som leder till det ikoniska brusljudet. Då bildsignalen uppfattas av tv:n på samma sätt, är utdatat flera bilder som bestod av slumpmässigt placerade vita och svarta rutor. Vitt brus

används mest för att lägga till detaljer till texturer. Fast vitt brus är inte särskilt användbart för terränggenerering, är det viktigt att förstå hur brus kan se ut i praktiken.

3.2 Perlinbrus

Perlinbrus skapades av Ken Perlin år 1985. Den är bland de äldsta och vanligaste brusfunktionerna som används inom PCG. Perlinbrus är en sorts lutningsbrus där algoritmen interpolerar punkternas värden med sina grannar [4]. Algoritmen börjar med att skapa ett rutnät med slumptalsvektorer för varje hörn. Sedan väljs en godtycklig punkt i varje ruta, varifrån algoritmen sedan kalkylerar en



Figur 2, a. Rutnät med slumptalsvektorer, b. En godtycklig punkt används för att räkna distansvektorer till varje hörn. Från [7].

distansvektor mot varje hörn i rutan. Skalärprodukten mellan de två vektorerna i varje hörn definierar lutningen. Slutligen interpolerar man alla värden från de närliggande rutorna för att skapa smidigare övergångar till grannpunkterna, se figur 2. Det skapar mönster som passar utmärkt för genereringen av till exempel moln eller berg. För slutprodukten, se figur 4.

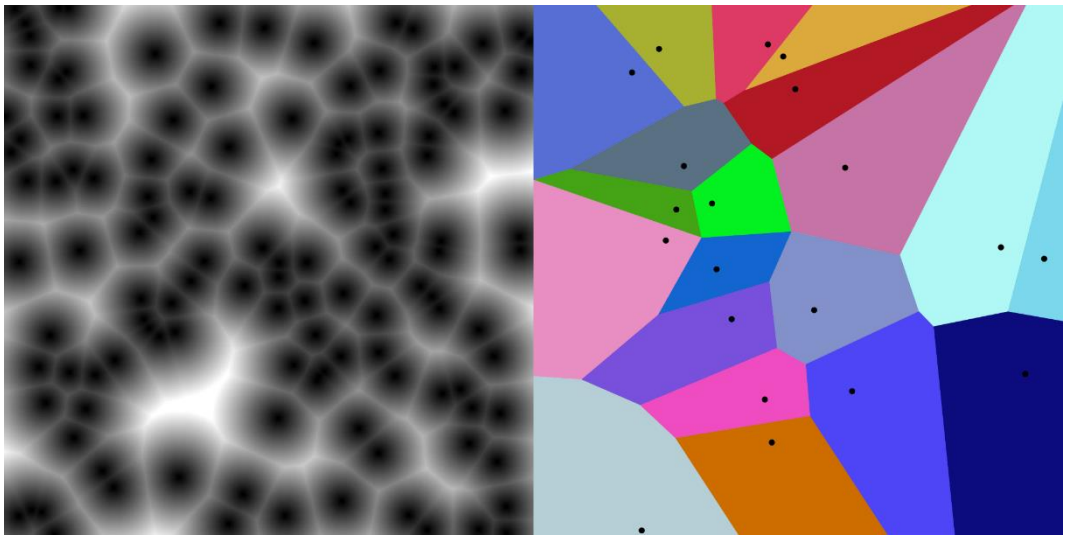
3.3 Simplexbrus

År 2001 skapade Perlin Simplexbrus som skulle förbättra hans ursprungliga Perlinbrus-algoritm. I stället för att alltid använda ett rutnät som Perlinbrus har, består ett Simplexnät av *n*-simplex, dvs. det enklaste upprepbara föremålet som fyller upp hela volymen för *n*-dimensioner. För två dimensioner är det en liksidig triangel, medan för tre dimensioner är det en tetraeder [9]. På detta sätt skulle

komplexiteten minska till $O(n^2)$ från Perlinbrusets $O(2^n)$. Han lyckades men Simplexbrus visade sig vara en för invecklad metod för de få förbättringarna som den erbjöd [7]. Utöver det, ansökte Perlin om patent för användningen av flerdimensionellt Simplexbrus. Problemen med tillgängligheten i kombination med att ursprungliga Perlinbruset var helt fritt att använda, ledde till att Simplexbrus inte helt ersatt Perlins ursprungliga algoritm.

3.4 Worleybrus

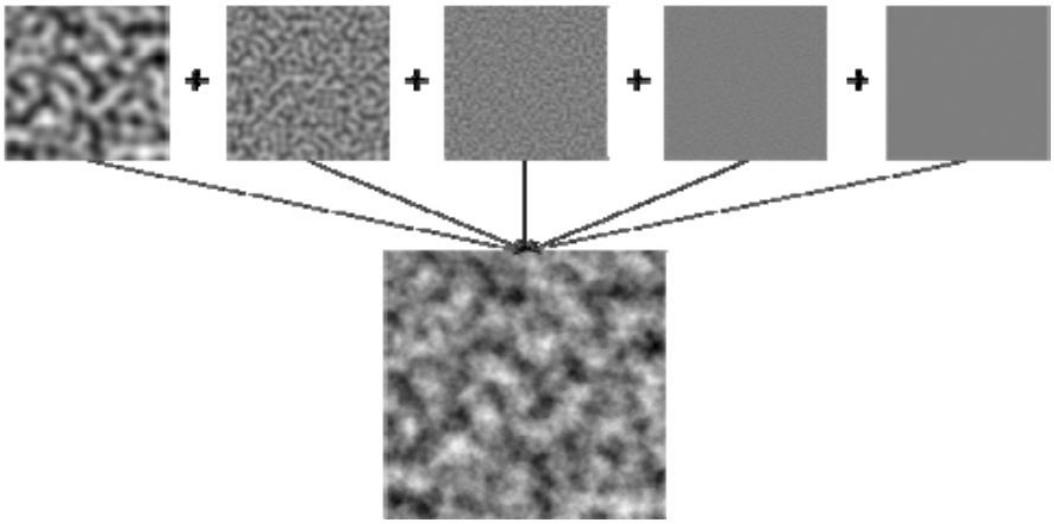
Worleybrus skapades av Steven Worley år 1996. Han ville skapa en ny grund för brus på samma sätt som Ken Perlin hade med hans Perlinbrus. Grunden för Worleybrus-algoritmen är att den slumpmässigt placerar ”egenskapspunkter” på brusfältet. Efter det väljer den en godtycklig position på fältet och en egenskapspunkt. Den minskar på avståndet tills den når ett läge där två eller flera punkters avstånd är lika. Där drar den då en ”cellvägg”. Dessa gränser dras enligt samma principer som i ett traditionellt Voronoi diagram, se figur 3. Lutningen av en position kan definieras på flera sätt, det enklaste är att färga dem enligt avståndet till närmaste egenskapspunkt. På detta sätt kan man skapa mönster med tydliga gränser. Worleybrus kan användas för att generera texturer som liknar stenplattor eller celler [8].



Figur 3, Worleybrus på vänster sida, Voronoi diagram med synliga egenskapspunkter på höger sida. Från bådas respektive artiklar på Wikipedia.

3.5 Fraktalbrus

Brusfunktioner behöver inte fungera isolerat från varandra. Med fraktalbrus (kallas även för turbulensalgoritmer) kan man addera olika frekvenser av brus för att modifiera resultatet, se figur 3. Desto mera frekvenser man kombinerar, ju mera detalj man får. Högre frekvenser (brussampel längre till höger i figur 3) ger subtilare detaljer, medan lägre frekvenser främst påverkar grundläggande strukturen. Med hjälp av dessa algoritmer kan man ge bruset mer komplexa mönster, till exempel träd, moln, marmor eller eld [7]. Det är värt att notera att fraktalbrus inte hör till andra fraktalbaserade metoder som behandlas i denna text.



Figur 4, en turbulensalgoritm adderar olika frekvenser av Perlinbrus för att öka på mängden detalj i bruset. Från [7].

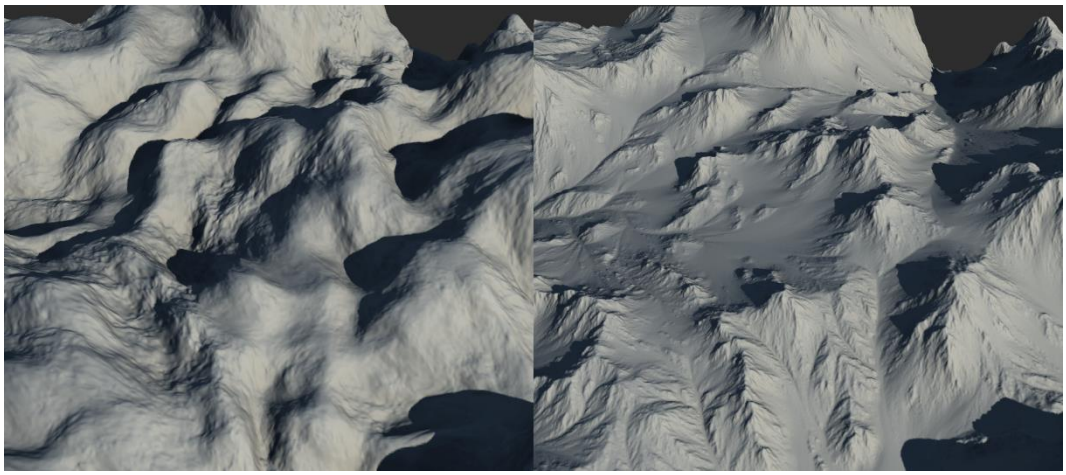
4. Fraktaler

5. Övriga metoder för att förbättra terrängen

De flesta brus- och fraktalbaserade terränggenereringsmetoder i sig själv är ganska primitiva. Man kommer inte att nå den höga kvalitén som moderna spel och speciellt filmer förväntas ha. Därför använder man flera andra metoder för att finslipa terrängen. I detta kapitel diskuteras metoder som inte är grundläggande för terränggenereringen, men spelar en viktig roll i att få det att se realistiskt ut.

5.1 Hydraulisk Erosion

Hydrauliska erosionsmodeller är algoritmer som simulerar regn och dess påverkan på terräng över länge perioder. Dessa algoritmer grundar sig på partikel- och vätskedynamiska modeller som naturligtvis skapar väldigt verklighetstrogna resultat.



Figur 5, terräng genererat med Perlinbrus. Bilden till vänster är före hydrauliska erosionsmodellen har använts, och högra bilden är efter. Bilderna är skapade med Gaea-terränggenereringsverktyget.

Grundidén bakom hydrauliska erosionsmodeller är att de simulerar hur vatten rinner samt hur det eroderar och flyttar partiklar i terrängen. Det vanligaste sättet är att simulera beteendet av enskilda vattendroppar. Algoritmen kan endast användas på en terräng som är färdigt genererad, dvs. det går inte att generera terrängen parallellt med denna algoritm. Se figur 5.

En regndroppe placeras på en godtycklig position i terrängen. Då börjar den med att flytta sig till den närmaste lägsta cellen med största höjdskillnaden. Regndroppen fortsätter att röra sig neråt enligt minsta motståndets väg. På vägen ner, kan den erodera och ”plocka upp” en partikel från terrängen. Då en del av terrängen eroderas, sänks höjdvärdet där det skedde. Det motsatta händer då regndroppen försvinner eller den når ett läge där alla närliggande celler har större höjdvärden. Denna process kan upprepas *x* antal gånger för att simulera olika mängder regn. Man kan justera parametrar som gravitation, friktion, ytspänning, markens hårdhet och andra fysiska krafter för att få olika resultat. Moderna

terränggenereringsapplikationer har ett stort utbud av parametrar som påverkar terrängen på olika sätt [3].

Ett annat sätt är att använda vätskesimulationer som simulerar flera partiklar samtidigt. Dessa används huvudsakligen för att generera åar, dalar, kanjoner och andra liknande egenskaper i naturen där relativt stora mängder vatten rör sig samtidigt [3]. Fast de ger oftast väldigt bra resultat, används de mindre för terränggenerering eftersom man har sämre kontroll över erosionen. På grund av det, måst man ofta köra algoritmen flera gånger före man får ett önskvärt resultat.

6. Referenser

[1] B. Mark, T. Berechet, T. Mahlmann, and J. Togelius, 'Procedural Generation of 3D Caves for Games on the GPU: Foundations of Digital Games', 2015. Avilable: https://lucris.lub.lu.se/ws/portalfiles/portal/6067634/5464988.pdf.

[2] A. Zafar, H. Mujtaba, and M. O. Beg, 'Search-based procedural content generation for GVG-LG', *Applied Soft Computing*, vol. 86, p. 105909, Jan. 2020. Available: https://www.sciencedirect.com/science/article/pii/S1568494619306908.

[3] H. T. Beyer, 'Implementation of a method for hydraulic erosion', Nov. 2015, [Online]. Available: https://www.firespark.de/resources/downloads/implementation%20of%20a%20methode%20for%20hydraulic%20erosion.pdf

[4] L. Ares *et al.*, 'State of the Art in Procedural Noise Functions', May 2010. http://www-sop.inria.fr/reves/Basilic/2010/LLCDDELPZ10/ (accessed Feb. 22, 2023).

[5] T. Sanford, 'Dynamic Voxel Based Terrain Generation', *Computer Science and Software Engineering*, Jun. 2015, [Online]. Available: https://digitalcommons.calpoly.edu/cscsp/47

[6] D. S. Ebert and F. K. Musgrave, *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2003. Available:

http://elibrary.lt/resursai/Leidiniai/Litfund/Lithfund_leidiniai/IT/Texturing.and.Modeling.-.A.Procedural.Approach.3rd.edition.eBook-LRN.pdf

[7] A. Tatarinov, 'Perlin noise in Real-time Computer Graphics', [Online]. Available: https://gc2011.graphicon.ru/html/2008/proceedings/English/S8/Paper_3.pdf

[8] S. Worley, 'A cellular texture basis function', in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, Not Known, 1996, pp. 291–294. Available: http://portal.acm.org/citation.cfm?doid=237170.237267

[9] S. Gustavson, 'Simplex noise demystified', Jan. 2005, [Online]. Available: https://www.researchgate.net/publication/216813608_Simplex_noise_demystified