Implementace algorithmu Jump Flood pro CUDA

Tomáš Weiss

ČVUT–FIT

weisstom@fit.cvut.cz

20. června 2024

1 Úvod

Tato zpráva pojednává o semestrální práci předmětu NI-GPU zaměřené na rychlé generování signed distance fields z heightmap. Na to byl použit algoritmus Jump Flood[2], který byl implementován na CPU i GPU.

2 Výpočet SDF

2.1 Definice SDF

Signed distance field je v počítačové grafice běžně textura, která v každém texelu ukládá vzdálenost k nejbližšímu bodu nějakého tvaru[1]. Tato textura má standartně rozměry stejné jako dimenze SDF. Nejčastěji se používá euklidovská vzdálenost.

https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-34-signed-distance-fields-using-single-pass-gpu

2.2 Naivní řešení

V našem případě se jedná o 3D euklidovskou vzdálenost a můžeme tak aplikovat různé metody pro výpočet vzdálenosti. Je možno například pro každý texel iterovat v pevně velkém sousedství, spočítat vzdálenost ke každému bodu pod povrchem heightmapy a uložit nejkratší vzdálenost.

Můžeme také použít upravený proces konvoluce se speciálním kernelem, díky čemuž je možno využít existujících knihoven pro konvoluci a urychlit tak výpočet.

Takovéto metody jsou ale pomalé a počet jejich operací rychle roste s velikostí SDF.

2.3 Jump flood

JFA je algoritmus, který má výpočetní složitost $O(\log n)$ za cenu výpočtu pouhé aproximace SDF. Funguje na opačném principu, místo hledání nejbližší vzdálenosti pro každý texel distribuuje nejbližší bod.

Algorithm 1 Jump flooding algoritmus

Require: N (rozměr obrázku)
  for $k \in \{\frac{N}{2}, \frac{N}{4}, \ldots, 1\}$ **do**
    for každý pixel p na (x, y) **do**
      for každého souseda q na $(x+i, y+j)$ kde $i, j \in \{-k, 0, k\}$ **do**
        if p je nedefinovaný a q je definovaný **then**
          Změň barvu p na barvu q
        else if p je definovaný a q je definovaný **then**
          if $dist(p, s) > dist(p, s')$ kde s a s' jsou seed pixely pro p a q **then**
            Změň barvu p na barvu q
          end if
        end if
      end for
    end for
  end for

3 Implementace

Algoritmus se skládá ze 3 typů průchodů. Na začátku se vykoná **seed**, který všem texelům, které jsou uvnitř tvaru, přidělí jako hodnotu jejich souřadnici. Poté se opakuje **step** (popsaný v 1), který hledá souřadnici nejbližšího seed texelu v postupně zmenšujících se krocích. Posledním průchodem je **distance**, který pro každý texel spočítá vzdálenost jeho pozice od jeho nejbližšího nalezeného seed texelu.

3.1 CUDA

Algoritmus potřebuje v paměti držet 2 kopie celé 3D matice hodnot a nemá žádnou stabilní lokalitu, tudíž byla použita globální paměť. Díky tomu, že každý texel pouze čte informace a zapisuje pouze do sebe, je pro každý krok použité mapování 1 vlákno na každou 3D souřadnici.

Pro **seed** se nejdříve přenese heightmapa na GPU a pak se vytvoří paměť pro celou 3D matici. Spustí se vlákno pro každou 3D souřadnici a to spočítá, zda je pod povrchem heightmapy.

Každá iterace **step** je samostatné spuštění kernelu, jelikož je potřeba globální synchronizace. Obsahuje vnořený cyklus, který prohledá sousedy ve vzdálenosti dané indexem iterace. Pro porovnání vzdáleností je používá jejich druhá mocnina, aby se eliminovalo použití odmocniny.

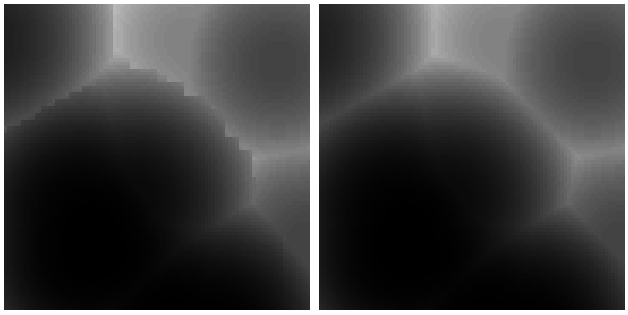
Krok **distance** používá funkci `norm3df` pro výpočet euklidovské vzdálenosti.

3.2 CPU

Implementace na CPU je prakticky identická GPU implementaci. I když tento algoritmus není adaptován na CPU, je díky nižší algoritmické komplexitě rychlejší než možné naivní CPU implementace.

4 Přesnost

Tento algoritmus počítá pouze aproximaci distance field, nemusí vždy správně distribuovat hodnoty. Toto lze vylepšit opakováním určitých iterací za cenu pomalejšího výpočtu[3]. V této implementaci je kromě klasické možnosti **JFA** ještě **1+JFA**, která před samotnou iterací **step** spustí jednu iteraci s radiusem 1. Toto zlepšuje šanci, že při následujících iterací texely narazí na správnou hodnotu. Dále je přidána možnost **1+JFA+2**, která dál rozšiřuje algoritmus o opakování iterací s radiusem 2 a 1 na konci, což zjemní chyby. Nakonec obsahuje $\mathbf{JFA^2}$, který iteruje dvakrát a zajišťuje téměř perfektní výsledky.



Obrázek 1: Porovnání samotného JFA (vlevo) a JFA^2

5 Výsledky

Jakých výsledků bylo dosaženo, co na ně melo vliv. Srovnání s očekáváním, *diskuze nad výsledky* – zvláště důležitá v případě, že něco vyšlo *divně*.

6 Závěr

Algoritmus se podařilo úspěšně adaptovat na CUDA hardware. Na CPU došlo k jasnému zrychlení, ale doba výpočtu roste rychle při vyšších rozlišeních. Toto by šlo alespoň částečně vylepšit využitím více jader, ale jelikož se práce zaměřovala na CUDA implementaci, byla CPU verze ponechána jednojádrová.

Výpočti na CUDA probíhají extrémně rychle i při vyšších rozlišeních a většímu počtu vrstev. Na dobu výpočtu nemá příliš velký vliv náročnost vstupních dat, což naznačuje, že je algoritmus rychlý a většinu doby zabírají okolní činnosti.

Reference

[1] Hubert Nguyen. *Gpu gems 3*. Addison-Wesley Professional, first edition, 2007.

[2] Guodong Rong and Tiow-Seng Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, page 109–116, New York, NY, USA, 2006. Association for Computing Machinery.

[3] Guodong Rong and Tiow-Seng Tan. Variants of jump flooding algorithm for computing discrete voronoi diagrams. In *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*, pages 176–181, 2007.

Tabulka 1: Testování implementace

		CPU				CUDA			
Heightmap	Rozlišení	JFA	1+JFA	1+JFA+2	JFA^2	JFA	1+JFA	1+JFA+2	JFA^2
mountain	128x128	0.15	0.18	0.25	0.29	0.08	0.09	0.09	0.10
	256x256	0.67	0.78	1.04	1.29	0.09	0.09	0.09	0.09
	512x512	2.92	3.38	4.41	5.62	0.11	0.10	0.10	0.10
ground	128x128	0.15	0.18	0.24	0.29	0.10	0.08	0.09	0.10
	256x256	0.67	0.79	1.04	1.29	0.10	0.08	0.09	0.09
	512x512	2.90	3.41	4.45	5.68	0.11	0.10	0.10	0.11

v sekundách, 15 vrstev, NVIDIA GeForce RTX 2070 Super