GEGEVENSSTRUCTUREN EN ALGORITMEN: IMAGE COMPOSITING ALGORITME

Liam Volckerick

MAY 10, 2018

KULEUVEN

Inhoudstafel

Inleiding

In dit practicum werden we geïntroduceerd tot het Image Compositing Algoritme. Dit algoritme laat ons toe meerdere afbeeldingen aan elkaar te binden. Het resultaat hiervan zal een mooie vloeiende panorama zijn.

Om dit resultaat te bekomen zullen we twee belangrijke methoden implementeren: seam en floodfill. Seam zal met behulp van Dijkstra's kortste pad algoritme en een afstandsfunctie de grens bepalen tussen de twee afbeeldingen. Deze functie zal meerdere grenzen berekenen en hieruit de optimale[1] teruggeven. Floodfill zal een nieuwe afbeelding creëren waarop hij de afbeeldingen zal invullen rekening houdend met de gevonden grens.

[1] Dit is de meest onzichtbare grens. De grens die het minste opvalt.

Afbeelding als een graaf

Gegeven onderstaande afbeeldingen:

(7,0,0) (0,1,0) (0,0,1)

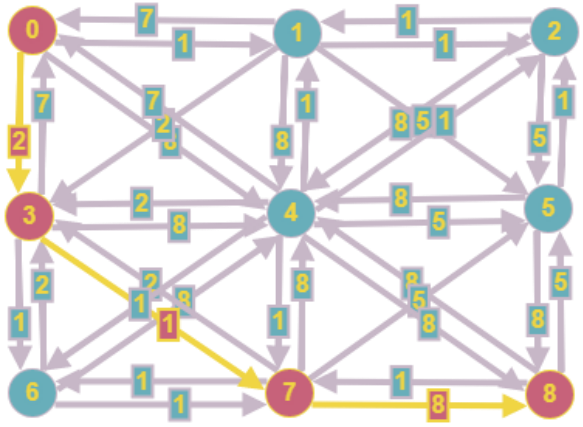
(2,0,0) (0,8,0) (0,0,5)

(1,0,0) (0,1,0) (0,0,8)

(0,0,0) (0,0,0) (0,0,0)

(0,0,0) (0,0,0) (0,0,0)

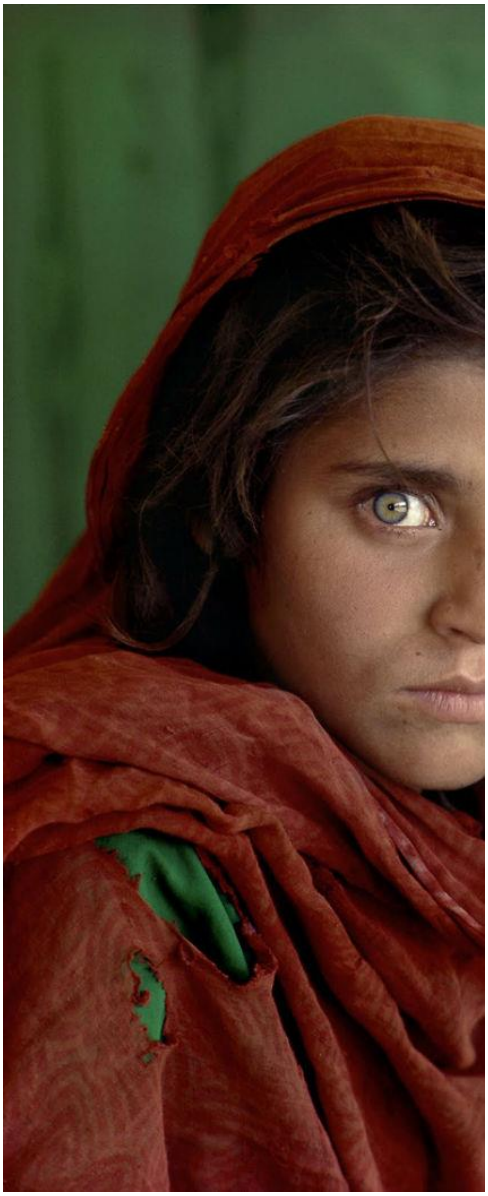
(0,0,0) (0,0,0) (0,0,0)



Op bovenstaande afbeelding zie je een representatieve graaf van de gegeven afbeelding uit vraag 1. De gewichten van elk pad zijn aangegeven en dusdanig het kortste pad berekend a.d.h.v. Dijkstra.

Alternatieve afstandsformule

De huidige afstandsformule bedraagt $\sqrt{r^2 + g^2 + b^2}$.



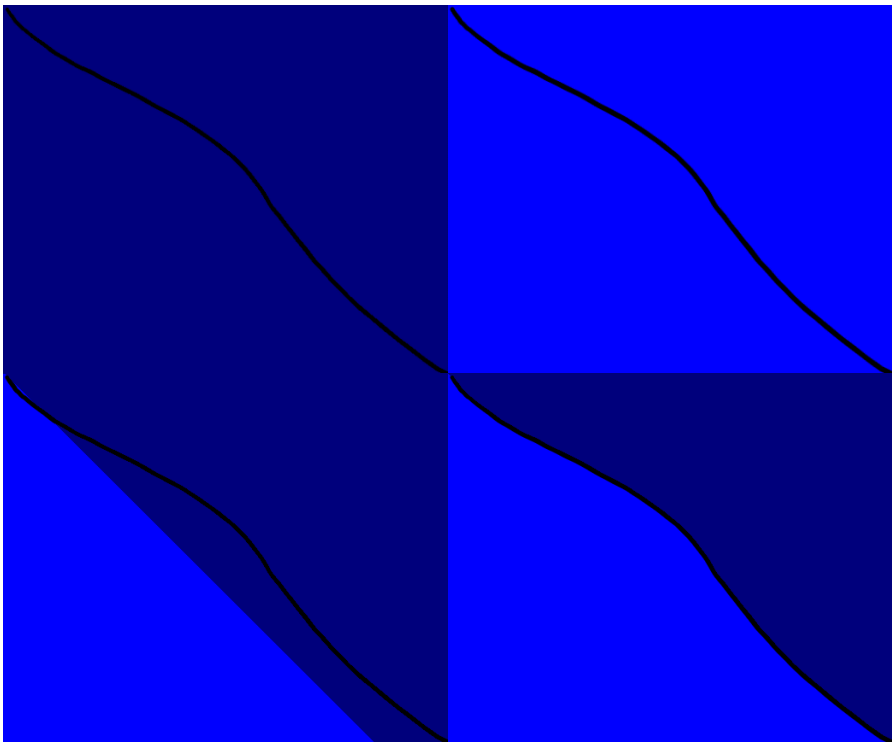
Figuur 1Portrait2.png

Figuur 2Portrait1.png

Indien we het algoritme laten uitvoeren op deze foto's via volgende commando: ant run -Dimg1=./images/portrait1.png -Dimg2=./images/portrait2.png -Doffsetx=270 krijgen we onderstaande resultaat. Op enkele details na zeker voldoende. Dit is in gebruik van de originele afstandsfunctie.

Figuur 3: Portrait-Out

Nu wordt er verondersteld hypothetisch een nieuwe afstandsformule te gebruiken: $\sqrt{r^2 + g^2}$. Nu we de blauwe component niet meer in rekening hoeven te brengen, welk effect heeft dat dan op het hele algoritme? In eerste instantie zouden we denken dat deze nieuwe afstandsfunctie een veel simpelere grens zal teruggeven doordat er een component wegvalt. Hoe groter de invloed van de blauwe component bij het bepalen van de grens, hoe korter het pad zal zijn m.b.v. deze nieuwe formule. De berekeningen tussen de onderlinge pixels zal ook versimpelen doordat de blauwe component wegvalt. Indien er een afbeeldingenpaar wordt gebruikt dat geen blauwige kleur bevat, zal er geen verschil zijn in het vinden van de grens tussen beide afbeeldingen. Zo zal er in de tegengestelde situatie, waar er zich heel veel blauw bevindt in het afbeeldingenpaar, geen goede grens meer gevonden worden. Dit komt doordat er geen correct pad kan berekend worden met behulp van Dijkstra's algoritme. Dit is te verklaren doordat elke graaf een evenwaardig gewicht zal hebben van 0. Met als gevolg dat alle mogelijke paden een evenwaardig gewicht hebben. Zo zie je in onderstaande voorbeeld het verschil in resultaat tussen beide afstandsformules.



Figuur 4: Twee bovenste afbeeldingen met kleurcode (0, 0, 124) en (0, 0, 255). Eerste resultaat(linksonder) is de afstandsformule zonder blauw. Rechts onderaan is het resultaat van het algoritme met afstandsformule incl. blauw.

Het gevonden pad kan ook zeer lang zijn indien we niet hoeven te stoppen wanneer de eind-node gevonden is. Doordat in een afbeeldingen paar met volledig blauwe kleur (0,0,255) elk pad een gelijk gewicht draagt en zal de priorityQueue elk pad aanschouwen als evenwaardig. In het geval van een volledig blauwe afbeeldingenpaar zou de grens theoretisch gezien steeds heen en weer bewegen doordat het steeds het beste pad opzoekt. Door het evenwaardige gewicht van elk pad, zou hij in worst case elke pixel zijn afgegaan op de afbeeldingen.

Tijdscomplexiteit

Bij een afbeelding met als grootte N*1 of 1*N zal de gevonden grens een lengte hebben van N-grootte. Dit volgt doordat er zich in deze gevallen alleen maar een rij of kolom van pixels bevindt met lengte N binnen de afbeelding.

De methode seam wordt als eerste opgeroepen, deze berekent de buren van elke pixel, met behulp van de hulpfunctie getNeighbours. Bij elke berekende buur zal er een grens worden toegevoegd aan de grafe. Indien we een afbeelding hebben met grootte NxN dan zouden we 8 buren berekenen voor elke pixel. Ongeacht welke dimensies de afbeelding mag hebben, heeft getNeighbours een complexiteit van 9. In bovenstaande geval zal deze maar 2 buren berekenen. Namelijk de bovenliggende en onderliggende of links en rechts.

Hierna zal het algoritme het kortste pad gaan berekenen. Deze heeft een complexiteit van $Elog_2(V)$ omwille van het gebruik van een priorityqueue. V stelt het aantal vertices voor en E het aantal edges(grenzen). In bovenstaande geval hebben we $E = 2(N-1)$ en $V = N$ wat dus resulteert in een complexiteit van $\sim 2(N-1)\ log_2(N)$. Dit komt doordat de beide eindpixels maar 1 buur hebben. We hoeven hier verder niets aan toe te voegen doordat ik een hashmap gebruik om mijn grens bij te houden. Deze werkt in lineaire tijd en mag dus verwaarloosd worden binnen de tijdscomplexiteit van bovenstaande afbeelding.

Floodfill zal elke pixel benaderen dat nog niet tot de grens behoort. Er worden dus niet meer dan N-pixels onderzocht omdat er eerst wordt nagekeken of de te bezoeken pixel reeds bezocht is. Zo ja, zal de floodfill stap overgeslagen worden voor betreffende pixel en zal de volgende pixel worden nagekeken. Indien we werken met bovenstaande afbeelding zal er geen enkele pixel worden benaderd. Dit doordat elke pixel reeds deel uitmaakt van de grens. Er zal dus geen bijkomende complexiteit komen in bovenstaande afbeelding door floodfill.

De totale complexiteit voor mijn Nx1 of 1xN afbeelding bedraagt dus $\sim 2(N-1)log_2(N)$

Indien we de complexiteit van Dijkstra voor een NxN-afbeelding willen benaderen horen we seam terug te bekijken. Binnen seam zal getNeighbours van bijna elke pixel nu 8 buren berekenen. Voor eender welke NxN afbeelding zal het aantal edges berekend worden door E = (N-2)*(N-2)*8 + (N-2)*4*5 + 4*3. V= NxN en dus zal de complexiteit van mijn Dijkstra implementatie (N-2)*(N-2)*8 + (N-2)*4*5 + 4*3 * log_2 (N^2) bedragen. Zoals je kan waarnemen is deze complexiteit enorm vergroot ten opzichte van een Nx1/1xN afbeelding. Het aantal buren dat wordt berekend in een NxN is veel groter dan bij een 1xN. Zo zal het Dijkstra algoritme trager verlopen voor een NxN afbeelding i.v.m. een 1xN/Nx1 afbeelding. Om het verschil duidelijker aan te tonen zal ik deze notatie vereenvoudigen.

$$E = 8N^2 - 32N + 32 + 20N - 40 + 12$$

$$E = \sim 8N^2$$

Dit toont duidelijk aan dat er zich haast een kwadratisch verschil bevindt in complexiteit tussen een NxN-afbeelding en een 1xN/Nx1-afbeelding. De complexiteit van een NxN-afbeelding bedraagt dus $\sim 16N^2log_2(N)$

Seam

Indien we onze seam willen voorkomen van complexe vormen aan te nemen zouden we enkele simpele aanpassingen kunnen maken aan het programma. Zo zou ik bij het berekenen van de buren richtingsgericht bepalen welke buren worden berekenend en welke nooit aan bod komen. Enkel de buren in rechtse en/of onderliggende richting worden berekend. Dusdanig creëren we een pad dat niet naar boven kan lopen noch naar links.

Langste niet-cyclische pad

Indien we het langste niet-cyclische pad nemen om dusdanig de grens te bepalen tussen meerdere afbeeldingen, zal de grens bestaan uit de hele afbeelding. Dit komt doordat elke pixel zal worden toegevoegd aan het pad. Het visuele resultaat zal nu wel afhangen van de implementatie van het algoritme en welke afbeeldingen we samenvoegen. Op de wijze die ons werd aangelegd zal de seam worden opgevuld door de tweede afbeelding. Indien deze grens nu over de hele afbeelding loopt zal het eindresultaat ons afbeelding 2 teruggeven. Dit resultaat wordt echter alleen bekomen indien de afbeeldingen van gelijke grootte zijn en geen offsetten worden toegepast.

Enkele andere voorbeelden

Rivier:

Nike: (m.b.v. -Doffsetx=365)