Gegevensstructuren en Algoritmen: Practicum 3

Robbe Degrève, r0662454

13 mei 2017

Inleiding

Het Image Compositing algoritme is een algoritme dat twee afbeeldingen zal samenvoegen. Zo worden er meerdere varianten gebruikt om meerdere afbeeldingen aan elkaar te hechten tot een panoramafoto. Hierbij is het belangrijk dat de overgang tussen de afbeeldingen zo weining mogelijk opvalt.

Om dit te verwezelijken hebben we 2 algoritmes nodig: seam() en floodfill().
Seam() zal met behulp van Dijkstra's kortste pad algoritme en een afstandsfunctie de grens bepalen tussen de twee afbeeldingen zodat de grens zo weinig mogelijk opvalt.
Floodfill() zal dan de nieuwe afbeelding invullen rekening houdend met de grens. De afbeeldingen worden dus aan elkaar gehecht.

1 Afbeeldingen als een graaf

Gegeven zijn onderstaande afbeeldingen met hun RGB-waarden.

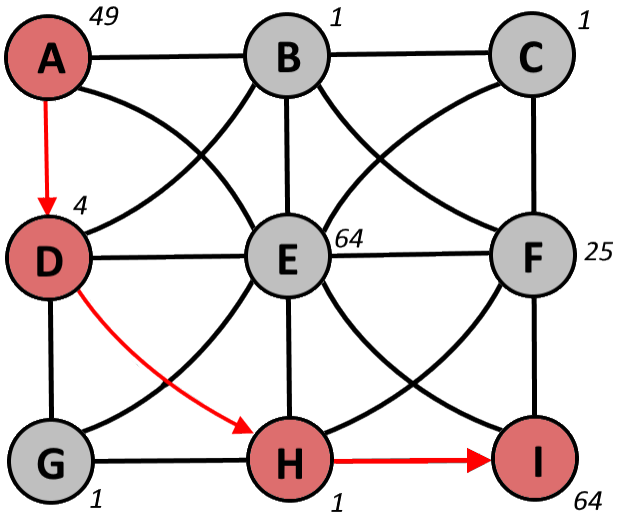
(7,0,0)	(0,1,0)	(0,0,1)
(2,0,0)	(0,8,0)	(0,0,5)
(1,0,0)	(0,1,0)	(0,0,8)

(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)

We geven iedere pixel van de resulterende afbeelding een naam om de grafe te verduidelijken. Deze namen en de *pixeldistance* tussen de twee afbeeldingen vind je terug in volgende tabel.

A - 49	B - 1	C - 1
D - 4	E - 64	F - 25
G - 1	H - 1	I - 64

Hierbij is de totale kost gelijk aan de som van de *pixeldistances* van de pixels op het pad. Het korste pad, aangeduid in het rood, heeft dus een kost gelijk aan $49 + 4 + 1 + 64 = 118$.



Figuur 1: Grafe met het kortste pad aangeduid

2 Alternatieve Pixeldistance-formule

We nemen nu een nieuwe afstandsfunctie $\sqrt{r^2 + g^2}$ i.p.v. de originele functie $\sqrt{r^2 + g^2 + b^2}$.

De blauwe component wordt nu dus niet meer bekeken bij het berekenen van de afstand tussen twee pixels. Hierdoor zal bijna iedere afbeelding in de praktijk een snellere uitvoeringstijd kennen, omdat er minder berekeningen gebeuren. Hoe meer blauw er aanwezig is in een afbeelding, of beter, hoe meer doorslaggevend blauw is voor de grens tussen twee afbeeldingen, hoe simpeler de grens er zal uitzien als we de nieuwe afstandsfunctie gebruiken. De grens wordt dus versimpeld en de afbeelding zal er anders uitzien.

In een afbeeldingpaar zonder de kleur blauw zal deze nieuwe functie geen effect hebben op de grens. Anderzijds zal het programma geen goede grens vinden tussen twee afbeeldingen waar enkel de kleur blauw in voorkomt. In dit geval zal het de korste weg in aantal pixels aanduiden als grens. Dit komt omdat in zo'n afbeeldingenpaar de kost van een pixel altijd gelijk is aan 0, en dus is het gewicht van eenderwelk pad 0. Hierdoor gaat het algoritme van Dijkstra zoeken naar een kortste pad in aantal pixels, omdat het stopt zodra het de eind-node heeft gevonden.

Het kan echter ook zijn dat dit pad zeer lang is, dit als we niet de variant van Dijkstra nemen die stopt als de eind- node is gevonden. Omdat het gewicht van ieder pad gelijk is aan 0, zal de PriorityQueue alle paden als "even lang"zien, alhoewel dat niet altijd zo is. Zo kan het dus ook zijn dat een bepaald (puur blauw) afbeeldingenpaar een zeer lange uitvoeringstijd heeft omdat de grens voortdurend heen en weer gaat.

3 Tijdscomplexiteit

Bij een afbeelding met als grootte N*1 of 1*N zal de seam de volledige afbeelding innemen. De lengte van dat pad is dus N.

Eerst wordt de methode *seam()* opgeroepen, die voor iedere pixel de buren ervan een nieuwe edge toevoegen aan de grafe. In dit geval zijn er maar 2 buren, dus we hebben al $2 * N$. Daarna moet Dijkstra, met een tijdscomplexiteit van $log(V) * (E + V)$, het korste pad bepalen. We hebben $E = 2 * N$ edges en $V = N$ vertices, dus we bekomen $log(N) * (3N)$. Deze seam wordt ook nog aan een lijst toegevoegd. Aangezien er N edges op het pad liggen, is de tijdscomplexiteit van *seam()* gelijk aan $N + 3N * log(N)$ of $\sim 3N * log(N)$.

Bij *floodfill()* wordt iedere pixel die niet op de seam ligt maar 1 enkele keer bezocht, dus de tijdscomplexiteit van *floodfill()* is $N - N = 0$, aangezien iedere pixel deel is van de seam.

Samen is de tijdscomplexiteit dus gelijk aan de tijdscomplexiteit van *seam()*,
namelijk $\sim 3N * log(N)$.

Als we dit nu toepassen op een afbeelding met als grootte NxN, krijgen we het volgende.
Voor *seam()*: Iedere pixel heeft 8 buren, dus $E = 8 * N^2$ edges en $V = N^2$ vertices. De tijdscomplexiteit voor Dijkstra is dus $18 * N^2 * log(N)$.
Voor *floodfill()* krijgen we een tijdscomplexiteit van $\sim N^2$.
Als we deze nu combineren krijgen we in totaal een tijdscomplexiteit van $N^2 + 18 * N^2 * log(N)$ of $\sim 18 * N^2 * log(N)$.

Dijkstra zal dus even snel zijn in functie van het aantal pixels als de afbeelding NxN groot is ipv 1xN of Nx1.

4 Seam en complexe vormen

Om te voorkomen dat de seam complexere vormen aanneemt kunnen we een simpele aanpassing maken. Voor de volgende node te vinden in het algoritme van Dijkstra gaan we kijken naar de buren van een bepaalde node. Hier kunnen we onze aanpassing maken door enkel de buren die zich rechts en/of onder die node bevinden te bekijken. We maken dus een gerichte graaf waarbij alle bogen naar rechts en/of naar onder gericht zijn, i.p.v. naar alle buren van een pixel.

5 Langste niet-cyclische pad

Als we het langste niet-cyclishe pad nemen als de grens tussen de twee afbeeldingen in plaats van het kortste pad, dan zal de grens de hele afbeelding innemen. Het visuele resultaat hangt nu dus af van het algoritme dat de afbeeldingen samenbrengt m.b.v. het masker. Dat masker bestaat uit enkel de grens en in onze implementatie wordt de grens vervangen met de tweede afbeelding. Het resultaat zal dus de originele tweede afbeelding zijn als de twee afbeeldingen even groot zijn en er geen offsets gebruikt zijn.

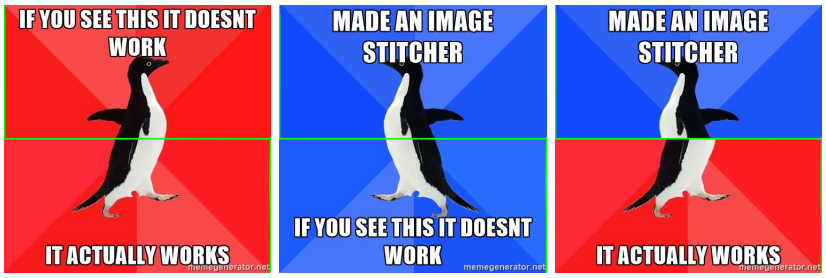
6 Extra voorbeelden

Onderstaande afbeeldingen zijn nog enkele extra, grappige afbeeldingen van de resultaten van het programma.

In dit eerste resultaat is het duidelijk dat een verschillend gekleurde lijn duidelijk nodig is.

Figuur 2: meme1.png, meme2.png en meme-output.png

Die lijn heb is hier in het groen toegevoegd.



Figuur 3: meme1a.png, meme2a.png en meme-a-output.png