

1. Meetrapport titel

1.1. Namen en datum

Lars Versteeg, Youri Mulder 04/01/2019

1.2. Doel

Wij gaan de standaard implementatie en onze Canny implementatie vergelijken op visuele aspecten zoals

- Dikte van de lijnen.
- Nauwkeurigheid van de randen.
- Vergelijking van de threshold waardes(bij Canny) (*1, 0.25, *0.5)
- Guassian sigma (1, 1.5, 2)

Ook willen wij in ons meetrapport de snelheid van het algoritme meenemen.

- Vergelijking van de snelheid van het volledige algoritme.

1.3. Hypothese

Wij verwachten dat onze implementatie dunnere randen levert en er beter uit komt te zien voor het menselijk oog. De tussenholding gaat veel verschil maken met de randen die wel of niet zichtbaar worden. De sigma gaat ervoor zorgen dat sommige randen zo erg worden vervaagd dat sommige randen niet meer op de juiste plek komen te liggen.

Wat betreft de snelheid zal onze implementatie ver achter blijven op de standaard implementatie. Het gebruik van verschillende waardes zal minimaal uitmaken. Dit gaat zo een klein verschil maken dat het niet meer te testen is. De computer kan met andere dingen bezig zijn die de metingen kunnen. Wij verwachten dat onze implementatie slomer is, omdat canny meer stappen heeft die hij moet aflopen voordat hij tot het uiteindelijke plaatje komt. Als je alleen een kernel zoals laplacian eroverheen laat gaan is dit een stuk sneller.

Bij thresholding wordt verwacht dat dit ongeveer dezelfde tijd in beslag neemt, omdat hier niet veel hoeft te gebeuren. Het enige wat hier gebeurt is de intensiteit omdraaien.

1.4. Werkwijze

Om de snelheid te testen hebben wij een clock gebruikt die zich bevindt in de standaard library chrono.

Voor de functie aanroep komt dit:

```
auto start = std::chrono::steady_clock::now();
```

Na de functie aanroep komt dit:

```
auto end = std::chrono::steady_clock::now();
```

```
std::cout << "Edge detection: "
```

```
    << std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count()
```

```
    << " milliseconds\n";
```

De rest van de factoren waar wij op testen kunnen we pas achteraf doen. Dit doen wij door een vakkundig team te laten beoordelen.

1.5. Resultaten

Zie tabellen aan onderzijde van het document.

1.6. Verwerking

De snelheid van de algoritme en de verschillen daarin kunnen we vergelijken voor het procentuele verschil te berekenen en het absolute verschil.

Procentuele verschil

$\text{verschil} = ((\text{waarde}_1 - \text{waarde}_2) / \text{waarde}_2) * 100$

Absolute verandering

$\text{verschil} = \text{hoogste_waarde} - \text{laagste_waarde}$

Dit zullen we doen voor onze eigen implementatie vergelijken met die van de standaard vergelijking

De andere aspecten waarop wij testen zijn afhankelijk van een jury. Hierin wordt meegenomen wat de gebruikte waardes van threshold en sigma doen met de plaatjes.

1.7. Conclusie

Snelheid

De standaard implementatie is sneller op het gebied van randen detecteren, maar slomer op het gebied van de thresholding. Dit komt doordat onze thresholding binnen de aanroep van randen detecteren valt. Het detecteren van randen duurt aanzienlijk veel langer. Wij verwachten dat dit komt, omdat canny überhaupt al langer duurt en onze implementatie niet optimaal is. Wij maken gebruik van 2D vectoren, dit duurt langer om de pixels te benaderen dan bij een 1D array. Onze implementatie is dus minder geschikt voor het real-time updaten dan de standaard implementatie wegens de snelheid.

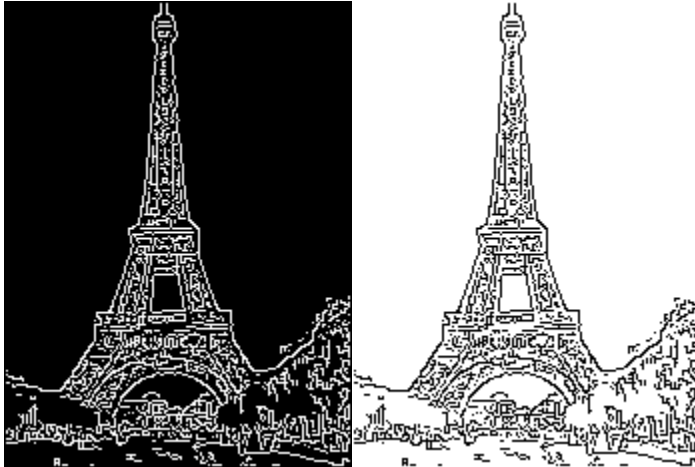
Threshold

Het verschil in de threshold waardes is meteen duidelijk. Bij de threshold waarde 0.25 kan je nog veel detail in het plaatje herkennen. Bij 0.5 zie je een stuk minder detail, maar het is nog duidelijk zichtbaar dat het een kind is. Sommige randen zijn verdwenen, maar de belangrijkste randen blijven bestaan. Bij de laatste waarde 1 wordt het zichtbaar dat zelfs belangrijke randen verdwijnen zoals neuzen, lichaamsdelen en bomenstammen.

Sigma

Zoals te zien heeft een hogere sigma dan 1 weinig impact op de afbeelding. Daarom hebben wij dit nogmaals getest met een lagere sigma. Dan zie je meteen verschil, omdat de vervaging een stuk minder is. De gebruikte instellingen zijn:

`double sigma = 0.5; double thresholdFactor = 0.25;`



Het is duidelijk dat een lagere sigma wel verschil maakt. Als je dit vergelijkt met het plaatje met dezelfde threshold zie je dat deze een stuk meer detail heeft in bij de bomen aan de rechterkant en de Eiffel toren zelf.

Nauwkeurigheid van de randen

De nauwkeurigheid verschilt heel erg per instelling. Zo vinden wij dat bij een threshold factor van 1 alleen de belangrijkste randen zoals het contour van persoon of het gebouw te zien is. Soms valt er zelfs een stukje van dit contour weg. Wij vinden deze instelling dus niet optimaal.

Een betere optie vinden wij een threshold van 0,5. Hierbij zie je duidelijk wat de randen moeten voorstellen en neem je het grootste detail weg. Deze threshold vinden wij het mooiste van de getesten thresholds.

Met een threshold van 0.25. komt er teveel detail bij. Hierdoor begint het te lijken op ruis. Bekijk bijvoorbeeld de plaatjes van het kind dan zie je ook lijnen in zijn wangen. Wij vinden dat dat niet hoort en dus deze threshold te laag ligt.

Dikte van de lijnen

Onze implementatie maakt gebruik van non-maximum suppression. Dit wordt meteen duidelijk als je naar de plaatjes kijkt. De randen zijn dunne lijntjes. Bij personen vinden wij dit er beter uitzien dan bij de standaard implementatie.

De standaard implementatie maakt zo te zien niet gebruik van een verdunning van de randen.

Wij vinden onze implementatie beter voor het oog mits je de goede thresholding toepast.

1.8. Evaluatie

Leg een verband tussen de getrokken conclusie en het doel van het experiment (en de hypothese). Ga daarbij ook in op bijvoorbeeld de meetonzekerheid als gevolg van de gebruikte meetmethoden of eventuele meetfouten.

Onze meetmethode zijn voor een deel gebaseerd op persoonlijke voorkeur, omdat wij geen harde waardes kunnen gebruiken naast thresholding en sigma is het lastig te zeggen welke daadwerkelijk beter is voor het oog. Als je een implementatie maakt waarmee de computer goed zijn werk moet kunnen doen kan je dit een stuk makkelijker bewijzen.



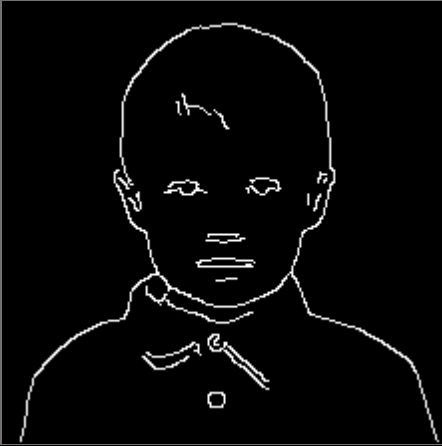
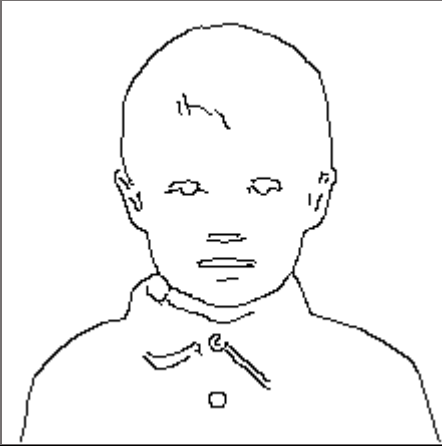
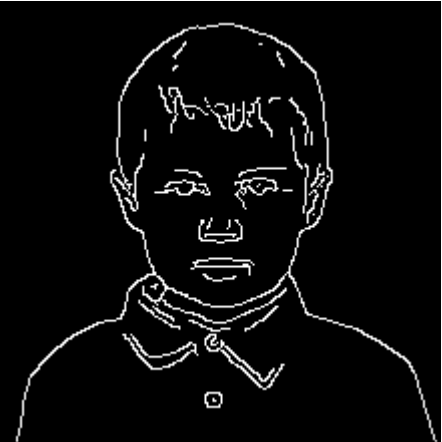
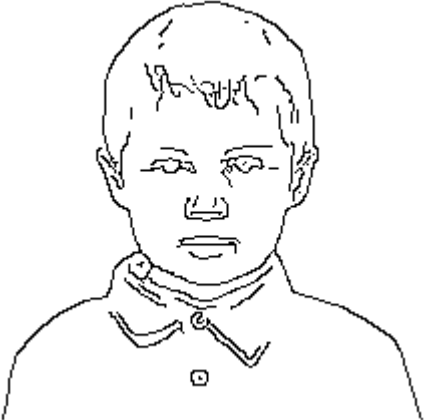
Tijdens het testen is gebleken dat wij beter andere sigma waardes hadden kunnen gebruiken. Hierdoor zien wij weinig verschil terug in de plaatjes door de verandering in de sigma. We hadden hier van tevoren beter naar moeten kijken.


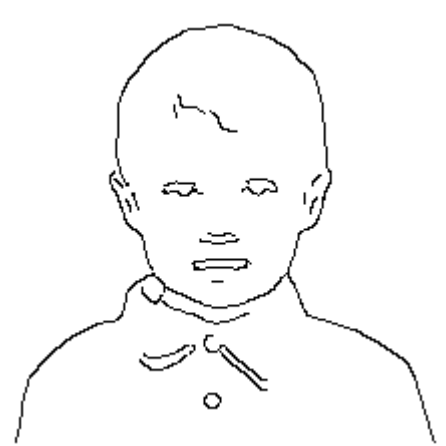
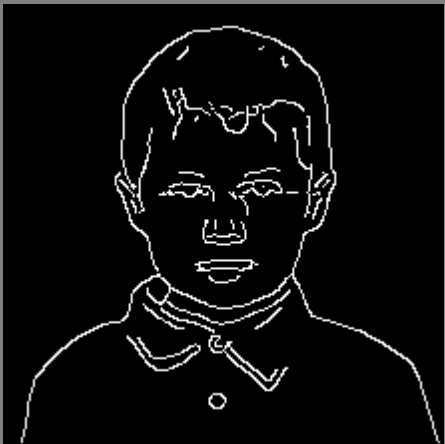
De verwachtingen van de hypothese zijn waar gemaakt.

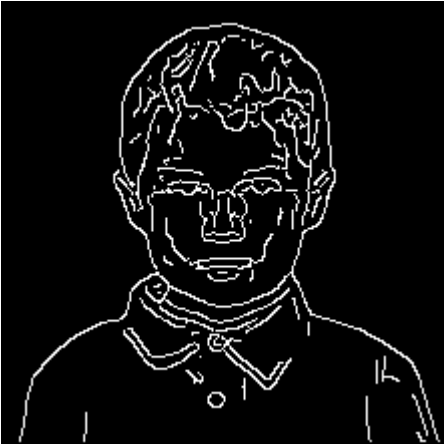

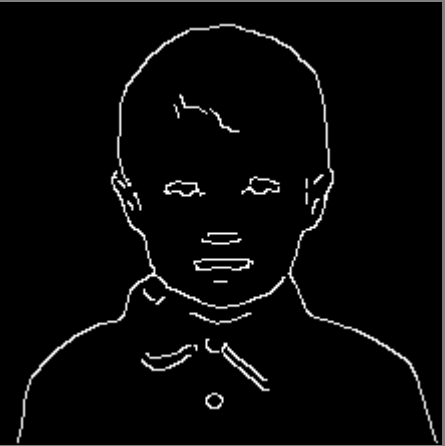
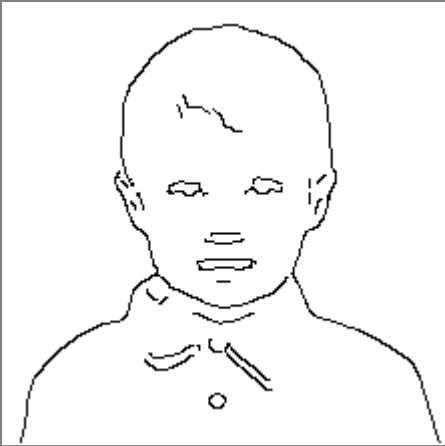
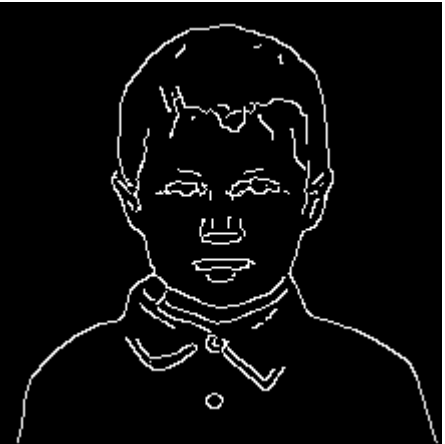

- Onze implementatie is slomer.
- Onze implementatie maakt geen verschil in snelheid door instel waardes.
- Onze implementatie heeft dunnere lijnen.
- Onze thresholding is sneller.

Het enige wat niet goed getest is kunnen worden met de gebruikte waardes is de sigma. Om erachter te komen of de sigma een verschil maakt moeten we lagere waardes gaan gebruiken dan we nu hebben gebruikt. Uit onze test met een lagere sigma die te zien is onder het kopje **sigma** in de conclusie zie je dat een lagere waarde ervoor zorgt dat er minder vervaagd wordt en dus een hoger detail in de uitvoer afbeelding geeft.



Implementatie	EdgeDetection(ms)	Threshold(ms)	Sigma	ThresholdFactor	Output EdgeDetection	Output Threshold	Detected
Standard	17	17	-	-			Volledig
Canny	561	12	1	1			
Canny	590	15	1	0.5			

Canny	574	14	1	0.25			
Canny	555	13	1.5	1			
Canny	565	13	1.5	0.5			

Canny	576	12	1.5	0.25			
Canny	506	14	2	1			
Canny	539	13	2	0.5			

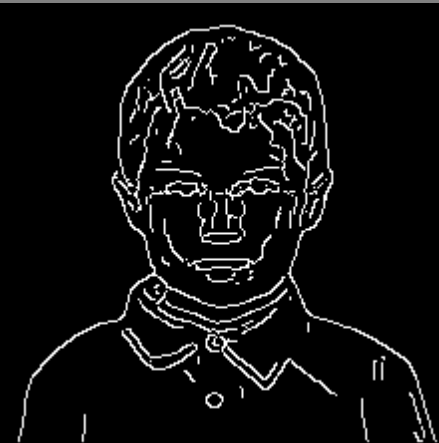
Canny

577

13

2

0.25



Searching for: Nose ends, and eye area's

Implementatie	EdgeDetection(ms)	Threshold(ms)	Sigma	ThresholdFactor	Output EdgeDetection	Output Threshold	Detected
Standard	22	17	-	-			
Canny	816	19	1	1			

							
Canny	813	18	1	0.5			
							
Canny	814	18	1	0.25			

Canny

810

16

1.5

1



Canny





807

19

1.5

0.5



							
Canny	790	17	1.5	0.25			
							
Canny	820	18	2	1			

Canny

814

17

2

0.5



Canny



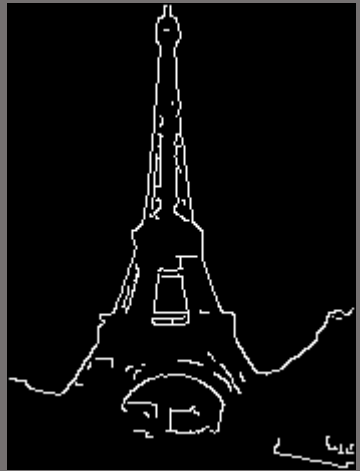
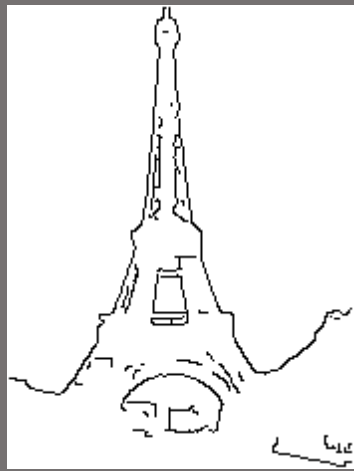
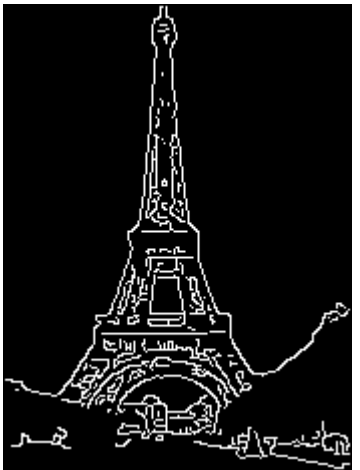

813

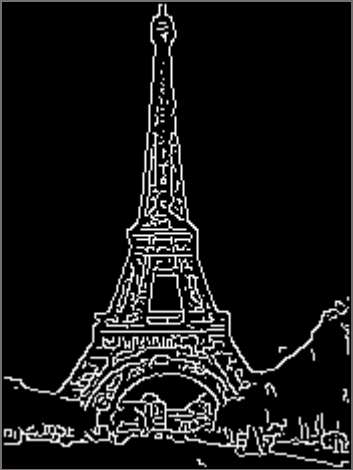
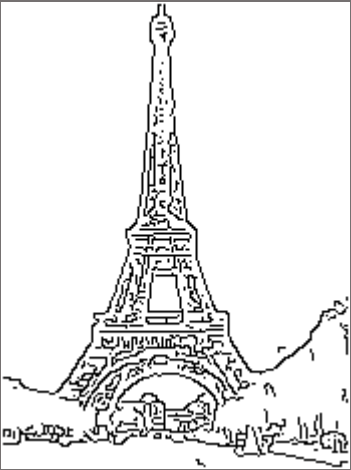
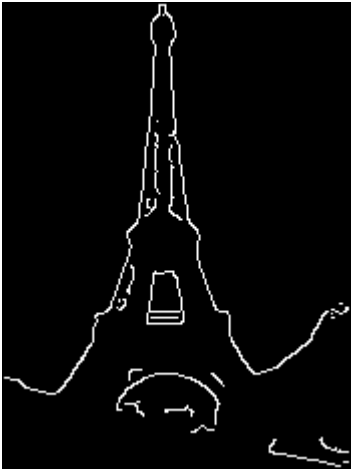
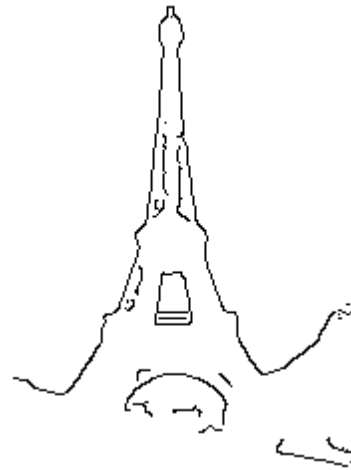
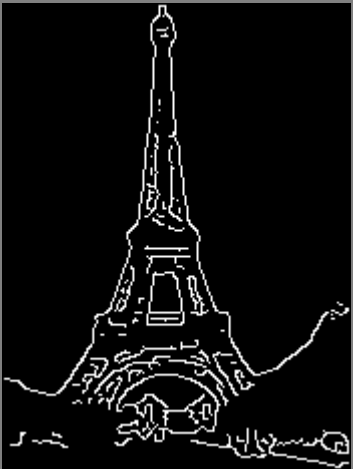
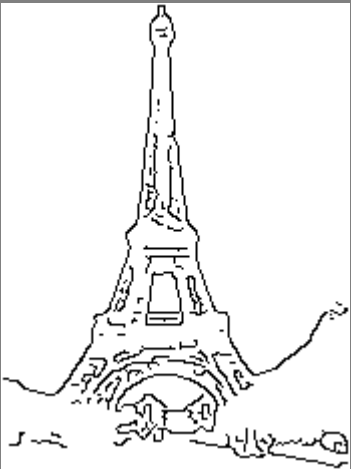
16

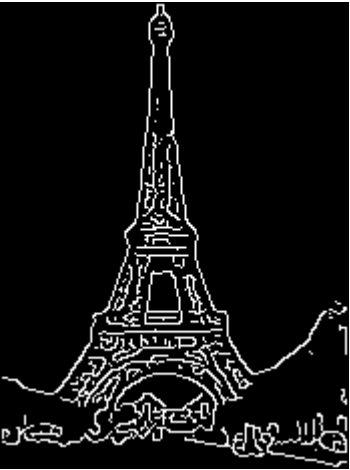

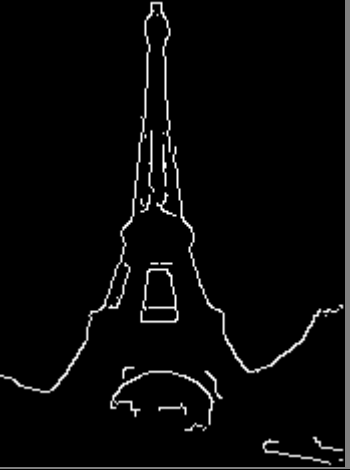
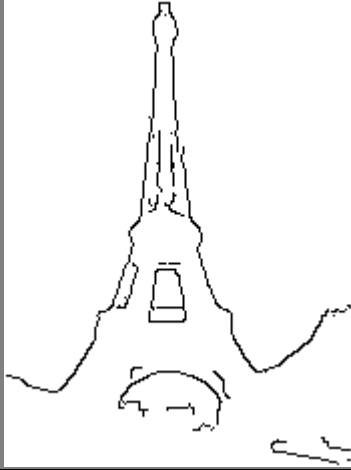
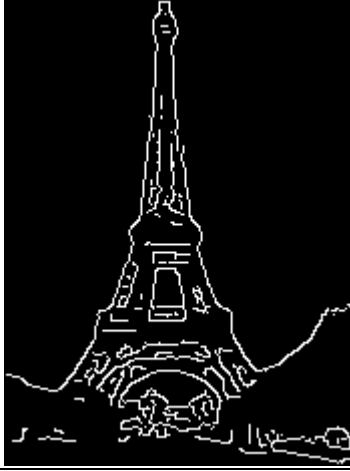
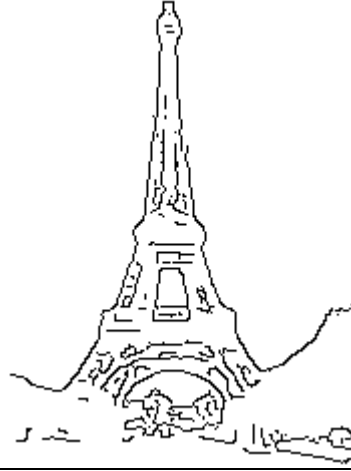
2

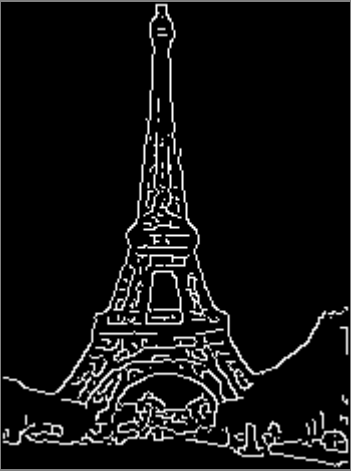

0.25



Implementatie	EdgeDetection(ms)	Threshold(ms)	Sigma	ThresholdFactor	Output EdgeDetection	Output Threshold	Detected
Standard	13	11	-	-			
Canny	477	10	1	1			
Canny	463	10	1	0.5			

Canny	469	11	1	0.25			
Canny	437	9	1.5	1			
Canny	416	9	1.5	0.5			

Canny	433	10	1.5	0.25			
Canny	462	10	2	1			
Canny	457	11	2	0.5			

Canny	435	10	2	0.25			
-------	-----	----	---	------	---	---	--