Experimentation project: Disparity map reconstruction from stereoscopic images

Sander van der Hurk*

August 29, 2014

Abstract

This report introduces the algorithm on Stereo Matching with Non-parametric Smoothness Priors in Feature Space by Smith et al.[9] and suggests an expansion of the algorithm by adding to the disparity map smoothing function a factor which takes into account the amount of edges crossed when smoothening the depth-map. The results show an improvement in the troubled areas of the original algorithm, but the expansion adds a factor $\sqrt{n}$ to the the $O(n^2)$ algorithm, increasing the runtime from 8 minutes to 22 minutes on a 480×360 image pair.

Introduction

Stereophotography is a way of simulating 3D images by combining two 2D images. Both images are captured at the same time with a small horizontal offset between the two lenses, as if each camera is a different eye of a person. When these two photos are viewed in such a way that the left eye sees one image, and the right eye sees the corresponding other image, the mind is tricked into believing the image has depth.

With the renewed interest in 3D imaging and film, the demand for stereoscopic 3D editing tools is rising[4]. The traditional editing tools do not take into account that after editing, the illusion of 3D should be kept intact. This is because the depth-information is only implied by the image, there is no real depth data stored with the image that a computer can use.

To explicitly get that depth data, stereomatching is used. See the schematic in Figure 1. We represent the photographed scene with scene points s_1 and

*s.e.vanderhurk@uu.nl — 3083942

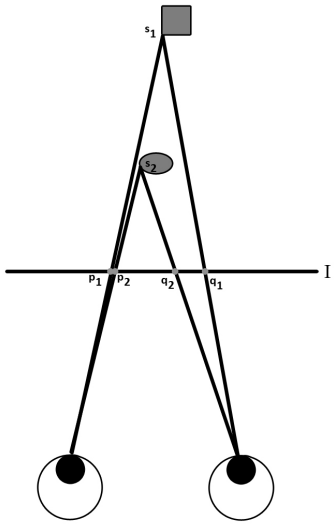


Figure 1: Schematic of a stereoscopic image

s_2. The points p_1 and p_2 are the points on the left image corresponding with the scene points s_1 and s_2. The points q_1 and q_2 are the corresponding points for the right image. Stereomatching is the name of taking 2 images of a stereoscopic 3D image, and extracting the depth information by determining which pixel p in the left image most likely represents the same scene point s as the pixel q does in the right image. The distance between p and q is called the disparity, and an image which shows in grayscale the distance between the point in the left image and that same point in the right image is called a disparity map. For an example of a disparity map, see Figure 3. The disparity is directly linked to the distance from the viewer to the object. The higher the disparity, the closer the object is to the viewer.

Once the disparity map has been constructed, this information can be used for example in photo slicing in the depth dimension [5]. To do this, it is of course important that the disparity map is very accurate.

Unfortunately, constructing a disparity map from two stereoscopic images is a non-trivial task [2]. Because two cameras are used to take the picture, the same point in a scene can have a different color in one camera compared to the other. This can be because of slight internal differences between the cameras, changes in reflected light due to a different perspective or calibration

problems. An even bigger problem is the possibility of occlusion in a single camera, causing the pixel of a scene-point to be only visible in one image and thereby not having a corresponding pixel in the other image.

The field of stereomatching can be roughly divided into two different parts: algorithms that have a short runtime but a high error in the resulting disparity map, and algorithms that have a long runtime but try to be pixel perfect about the disparity map [11][7]. The fast algorithms have most of their errors around jumps in disparity[11][7], and can therefor not be used for precise editing techniques.

Many precise stereomatching algorithms use a form of segmentation as a pre-processing step, which causes temporally inconsistent disparity maps when applied to video input [9]. The algorithm that we will be looking at, by Smith et al. [9], does not require such a segmentation and in stead uses minimum spanning trees to group the pixels with similar features, such as location and color, during the stereomatching so no hard decisions are made in the pre-processing step.

We will first explain the paper more in depth, after which we take a look at the runtime and results on different sized images.

After that, we propose an expansion of the feature vector used to build the minimum spanning trees, and show the runtime and results of the addition.

1 Related works

In 2002, Scharstein and Szeliski[7] set out to map the field of stereomatching algorithms, comparing over 30 algorithms on a certain set of stereoscopic images. This set, nowadays known as the Middlebury set, is still widely used in current papers as an indication of their performance. The corresponding website [8] still allows results to be send in.

In 2002, Kolgomorov and Zabih[3] used Markov Random Fields in stereomatching, and showed how to apply graph cutting to determine the minimum of the optimization function. Their basic energy function takes into account the correctness of a pair of disparity maps by reasoning if a disparity level is visible according to the other disparity map. This function will be used as a basis in the paper by Smith et al.[9], and we will discuss the function further on.

The stereomatching algorithm by Woodford et al. [12] is based on second-order priors. The results for piecewise planar images, such as the Venus from the Middlebury set, have excellent results. However, on the images with curved surfaces such as Cones it performs much worse.

Figure 2: Left Image from Smith et al. [9]

Figure 3: Disparity map for Figure 2

2 Stereo Matching with Nonparametric Smoothness Priors in Feature Space

In this section we will describe the algorithm as was proposed by Smith et al. [9] After that, we present the time tests run on the algorithm, as well as look at the output of the program. The results from our tests will be show less accurate disparity maps compared to the results in Smith et al. [9] This is due to the fact that they use 5 cameras for the reconstruction, whereas we will limit our use to two cameras to allow better comparison with other stereoscopic images. The difference in accuracy has also been noted by Zhu et al. [13], and the visual difference can be seen in Figures 4 and 5.

2.1 The algorithm

A novel formulation for stereo matching was proposed by Smith et al.[9], which is used in multiple papers (Luo et al.[6], Lo et al.[5]) as de facto disparity map algorithm. Their goal was to create a precise and consistent algorithm that can handle flat planes as well as high curvature depth regions, without resorting to segmentation.

The stereo matching algorithm can be viewed as an optimization problem where the sum of the local disparity maps should be minimized. The basic optimization formula $\Phi_{ph}(D_1, D_2)$ is used from Kolmogorov et al. [3]. Φ_{ph} measures the consistency between the disparity maps D_1 and D_2 of the stereo image pair I_1 and I_2 and is defined as

$$\Phi_{ph}(D_1, D_2) = \sum_{p \in I_1} \phi_{ph}(d_p, d_q)$$

Figure 4: Results from Smith et al. [9] as presented by Smith et al. [9]

Figure 5: Results from Smith et al. [9] as presented by Zhu et al. [13]

where p is the coordinates of a pixel in image I_1, q the coordinates of the pixel matching to pixel $I_1(p)$ according to disparity map $D1$ and therefor defined as $q = p + D1(p)$, $d_p = D_1(p)$ is the disparity for p and $d_q = D_2(q)$ the disparity for q.

If $d_p = d_q$ p and q are the same point in the scene according to the disparity maps. In that case $\phi_{ph}(d_p, d_q)$ returns a value $\rho_{ph} = min(0, \|c_p - c_q\|^2 - \tau_{ph})$, where $\tau_{ph} > 0$, $c_p = I_1(p)$ the intensity of p in image I_1 and $c_q = I_2(q)$ the intensity of q in image I_2.

If $d_p < d_q$, the point in the scene corresponding to p is occluded in image I_2. In this case disparity map cannot be checked in the point p, and $\phi_{ph}(d_p, d_q)$ returns 0.

If $d_p > d_q$, the point in the scene corresponding to p is closer than the point in the scene corresponding to q and should therefor occlude the point represented by $I_2(q)$. The disparity maps don't correspond to each other, and $\phi_{ph}(d_p, d_q)$ returns ∞.

Smith et al. add a smoothing factor to the minimization function. The idea behind this is as follows: Of the pixels are close together in the image, and the difference between the color is low, then it is reasonable to assume that the depth of those two pixels are close together.

To add this smoothing to the optimization formula the terms Φ_{sm} is introduced for disparity maps $D1$ and $D2$, resulting in the following function Φ:

$$\Phi = \Phi_{ph}(D_1, D_2) + \Phi_{sm}(D_1) + \Phi_{sm}(D_2)$$

where

$$\Phi_{sm}(D) = \sum_{p \in I} \phi_{sm}(d_p; \{d_q\}_{q \in \mathcal{N}_p})$$

Φ_{sm} regularizes the depth map based on the correlation between the pixel p and pixel q in p's neighborhood $\mathcal{N}_p$. Notice that Φ_{sm} works per individual image. In the neighborhood, pixels are viewed as feature vectors, and the image is viewed as a point cloud in feature space. A $5D$ vector $\mathbf{f} = [\mathbf{x}, \mathbf{c}]$ is used consisting of the pixel location $\mathbf{x} = [x, y]$ and color $\mathbf{c} = [r, g, b]$.

ϕ_{sm} is defined as

$$\phi_{sm}(d_p; \mathcal{N}_p) = -\lambda \log(\mathrm{P}(d|\mathbf{f}_p, \mathcal{N}_p))$$

where λ is the regularization coefficient, and $\mathrm{P}(d|\mathbf{f}_p, \mathcal{N}_p)$ returns the likeliness of disparity d_p given the feature vector $\mathbf{f}_p$ and neighborhood $\mathcal{N}_p$. Because Φ_{sm} will be minimized, the $-\log$ is used.

The conditional probability of d_p given $\mathbf{f}_p$ is modeled as

$$\mathrm{P}(d|\mathbf{f}_p, \mathcal{N}_p) = \frac{1}{\mathcal{N}_p} \sum_{q \in \mathcal{N}_p} w_{p,q} g_d(\frac{d - d_q}{\sigma_d})$$

where g_d is the kernel function for the disparity d, and σ_d the bandwidth associated with the disparity. The neighborhood weight $w_{p,q}$ is, according to the paper, modeled as

$$w_{p,q} = \frac{g_x(\frac{\mathbf{x}_p - \mathbf{x}_q}{\sigma_d}) g_c(\frac{\mathbf{c}_p - \mathbf{c}_q}{\sigma_c})}{\sum_{q' \in \mathcal{N}_p} g_x(\frac{\mathbf{x}_p - \mathbf{x}_{q'}}{\sigma_d}) g_c(\frac{\mathbf{c}_p - \mathbf{c}_{q'}}{\sigma_c})}$$

where g_x is the kernel function for the location $\mathbf{x}$, g_c the kernel function for the color $\mathbf{c}$, σ_x the bandwidth associated with the location and σ_c the bandwidth associated with the color. The Gaussian kernel function is chosen for $g_x(x)$ and $g_c(x)$ being

$$g_x(x) = g_c(x) = \exp(-\frac{x^2}{2\sigma^2})$$

making

$$w_{p,q} = \frac{\exp(-\frac{(\mathbf{x}_p - \mathbf{x}_q)^2}{\sigma_d^2 * 2\sigma^2}) \exp(-\frac{(\mathbf{c}_p - \mathbf{c}_q)^2}{\sigma_c^2 * 2\sigma^2})}{\sum_{q' \in \mathcal{N}_p} \exp(-\frac{(\mathbf{x}_p - \mathbf{x}_{q'})^2}{\sigma_d^2 * 2\sigma^2}) \exp(-\frac{(\mathbf{c}_p - \mathbf{c}_{q'})^2}{\sigma_c^2 * 2\sigma^2})}$$

which, when chosen $\sigma = 1$, is reduced to

$$w_{p,q} = \frac{\exp(-f_x((\mathbf{x}_p - \mathbf{x}_q)^2) - f_c((\mathbf{c}_p - \mathbf{c}_q)^2))}{\sum_{q' \in \mathcal{N}_p} \exp(-f_x((\mathbf{x}_p - \mathbf{x}_{q'})^2) - f_c((\mathbf{c}_p - \mathbf{c}_{q'})^2))}$$

where

$$f_x = \frac{1}{2\sigma_x^2}$$

$$f_c = \frac{1}{2\sigma_c^2}$$

To simplify the part of the formula we will later adjust, we will rewrite the formula as

$$w_{p,q} = \frac{\exp(-w'_{p,q})}{\sum_{q' \in \mathcal{N}_p} \exp(-w'_{p,q'})}$$

where

$$w'_{p,q} = f_x((\mathbf{x}_p - \mathbf{x}_q)^2) + f_c((\mathbf{c}_p - \mathbf{c}_q)^2)$$

The weight $w'_{p,q}$ is used to build sparse graphs using minimum spanning trees, which act as neighborhoods for the algorithm. We want to have $w'_{p,q}$ as close as possible to 1 when the disparity map should have a smooth transition between pixels p and q, and $w'_{p,q}$ close to 0 when the disparity map should not have a smooth transition.

2.2 Time measurements and output original algorithm

The program containing the algorithm from Smith et al. can be found on their own website [10]. It states that beside two images, a camera matrix is needed per image, and the algorithm can be run with only that information. An example image with resolution 480×360 is given, which we will use as ground truth for future references.

The camera matrix, which is not mentioned in the paper, consists of a 1×3 translation vector, a 3×3 rotation matrix, a 1×2 focal point vector, and a 1×2 principal point vector.

When adjusting the resolution of the image, the rotation and translation matrix can be maintained as is. The focal and principal point are changed by dividing the given values in the vector by 480 and multiplying by the width of the image.

The input file allows for many additional parameters, amongst which the radius for the neighborhood $\mathcal{N}_p$, the bandwidth of the color σ_c, and the bandwidth of the distance σ_x. A full list can be found in the readme by Smith et al. [10]

Since the original paper only states the runtime of a single configuration, we started with creating an overview of the algorithms runtime. Because we expect the runtime to be heavily dependent on the amount of pixels in the image we will test the program on a set of similar images with varying resolutions.

To do so, we resized the test image of the plant that came with the program, see Figure 2, from its original 480×360 pixels to resolutions k * $120 \times k * 90$ where $1 \leq k \leq 12$. To keep the camera matrices correct, we used the translation and rotation matrix from the original image, and linearly adjusted the focal point and principal point. All tests were run on an Intel 4930 CPU with 32GB cache memory.

2.2.1 NB-40

If no arguments are given to the program, the radius for the neighborhood $\mathcal{N}_p$ will get a value of 40 pixels, regardless of the resolution of the image. There is no explanation as to why this value is chosen other than "it seemed to work nice"[9]. Because of this value, we will refer to tests where we leave the algorithm with the minimal amount of arguments as the NB-40 configuration.

Runtime As can be seen in the graph in Figure 6, the NB-40 configuration has a runtime linear to the amount of pixels in the image, where the time ranges from 8 seconds on an 120×90 image, to 55 minutes on a 1920×1440 image.

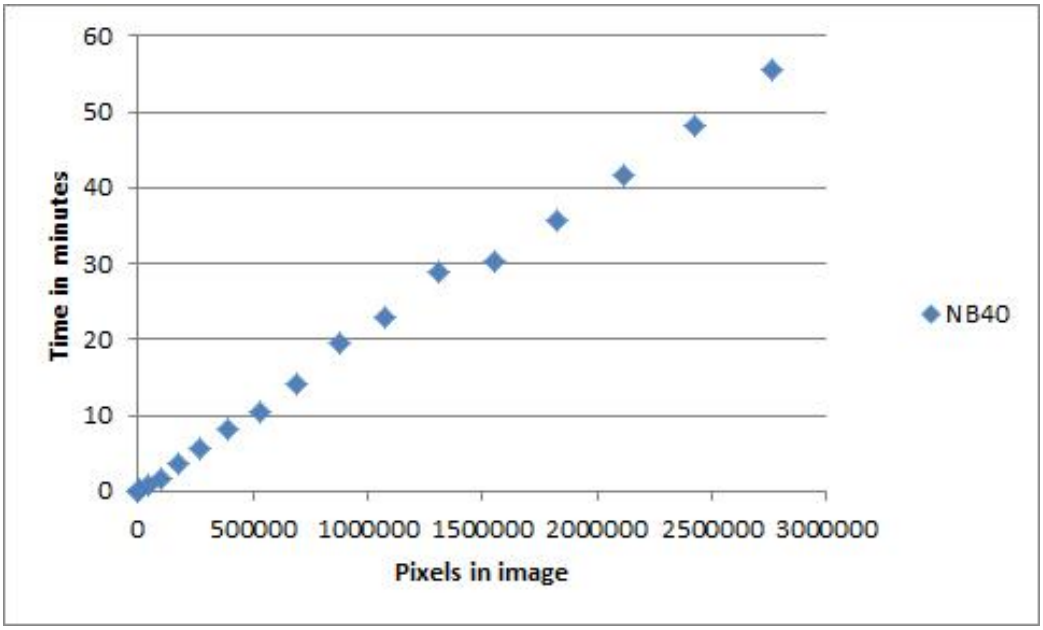


Figure 6: Average runtime of NB-40 configuration

Visual results When we look at the visual output of the NB-40 configuration (see Figure 7) we see that the disparity map is vastly different for each resolution. With the correct settings, the disparity map should roughly look the same regardless of resolution. These results can be explained by the choice of keeping the radius of $\mathcal{N}_p$ a constant. When the image resolution gets higher, the amount of pixels that a point in the scene is able to shift between the right and the left image also increases. Therefor, it is logical to increase the radius of $\mathcal{N}_p$ in which we seek our matching pixel along with the resolution.

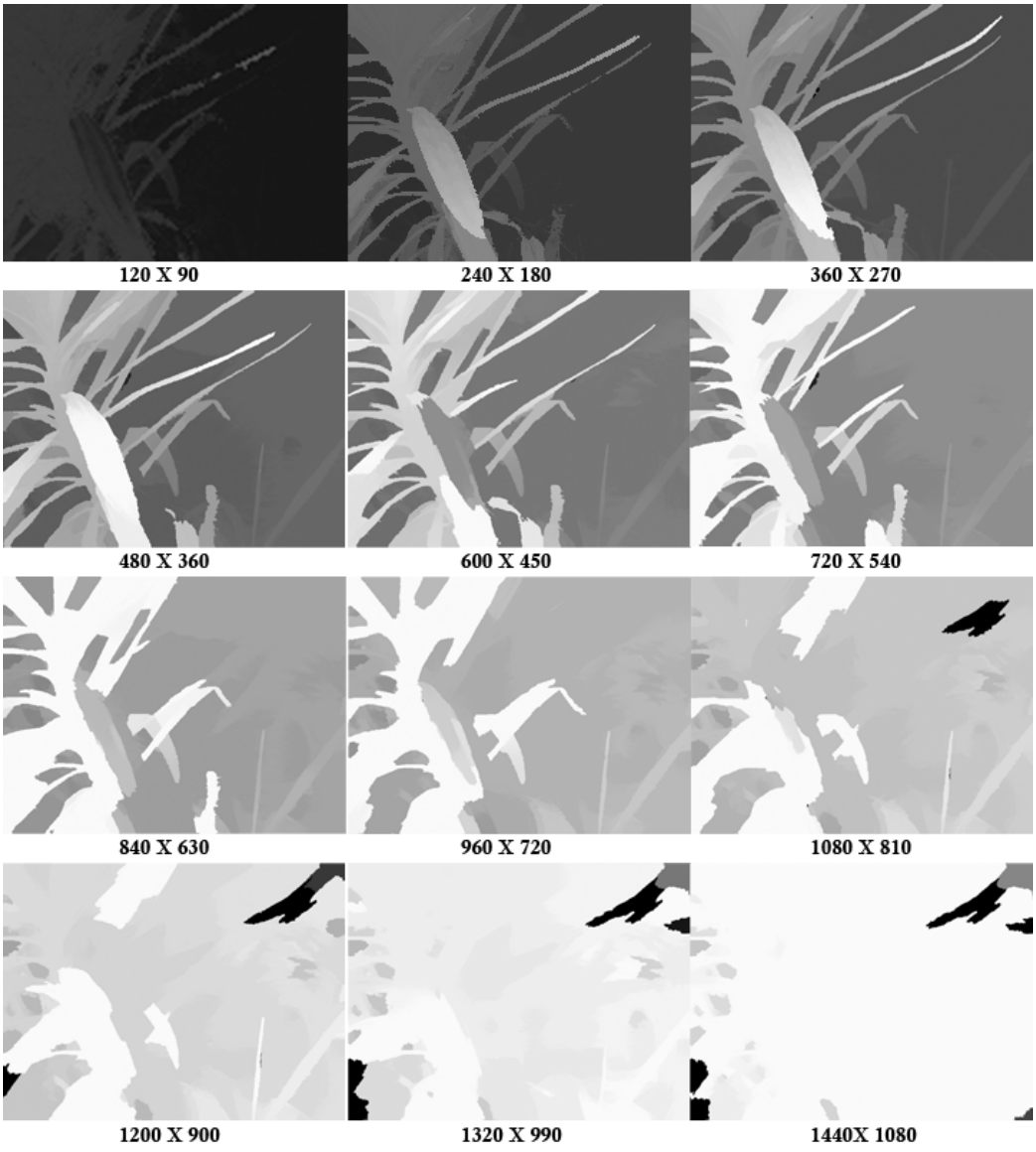


Figure 7: Results of disparity maps for the NB-40 configuration

2.2.2 NB-Var

To improve the results of NB-40 configuration, we ran a second set of tests where we change σ_x and radius for the neighborhood $\mathcal{N}_p$ along with the resolution of the image. Every value gets divided by 480 and multiplied by the width of the image. Because of the variable $\mathcal{N}_p$ radius, we will refer to the tests where we changed the radius along with the resolution as the NB-Var configuration.

The radius of $\mathcal{N}_p$ is changed linearly in the width of the image. Another variable we adjusted was σ_x, the bandwidth of the location in the weight formula w_p, q. The adjustment of σ_x was non-trivial. A logical choice would be to make sure that $\sigma_x * (2 * \texttt{radius}_{\mathcal{N}_p})^2$ stays the same regardless of resolution, as this would retain the weights of each relative distance in the image. However, the because of the division by the summation of all pixels in the algorithm does not divide by pixels in the radius of $\mathcal{N}_p$, making it seemingly impossible to feed the original algorithm with arguments in such a way that each resolution has the same disparity map.

We chose to adjust σ_x linearly with the width of the image, as we did with the other values. Because the runtime is independent of the chosen σ_x, the following run times can still be seen as relevant.

Runtime As Figures 8 and 9 show, the NB-Var configuration shows a quadratic runtime in the amount of pixels of the image. This was to be expected, as every pixel p will now look at $c * n$ pixels q where $0 < c \leq 1$ and n the amount of pixels in the image, making it an $\Omega(n * c * n) \subseteq \Omega(n^2)$ algorithm.

Visual results The visual results in Figure 10 show that the disparity map for varying resolutions is now more equal than before. For the resolutions 480×360 to 720×540 the results look similarly correct.

The resolutions below 480×360 show errors which could be explained by the lack of information in the image due to the resizing. The resolutions above 720×540 show similar errors as shown in Figure 5 by Zhu et al. [13]

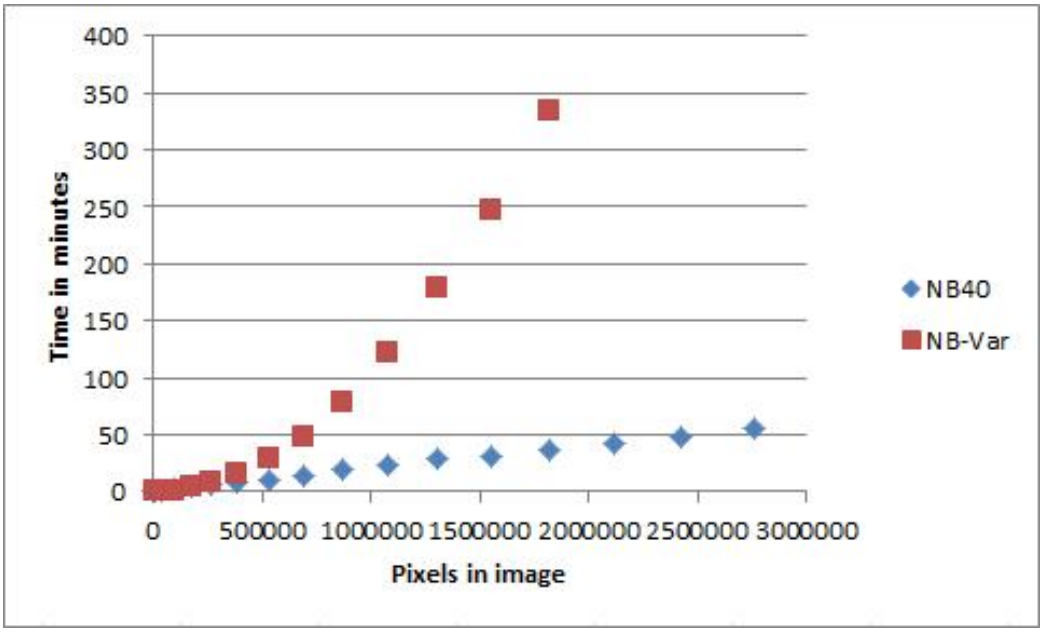


Figure 8: Average runtime of NB-40 configuration and NB-Var configuration, with the amount of pixels in the image on the x-axis.

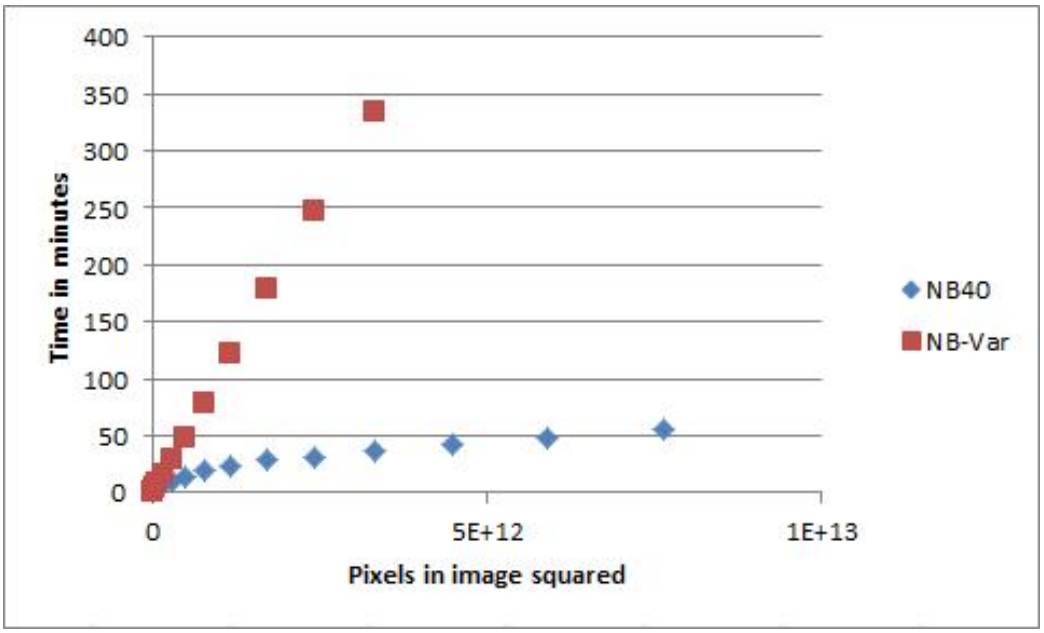


Figure 9: Average runtime of NB-40 configuration and NB-Var configuration, with the amount of pixels in the image squared (maximum amount of operations) on the x-axis.

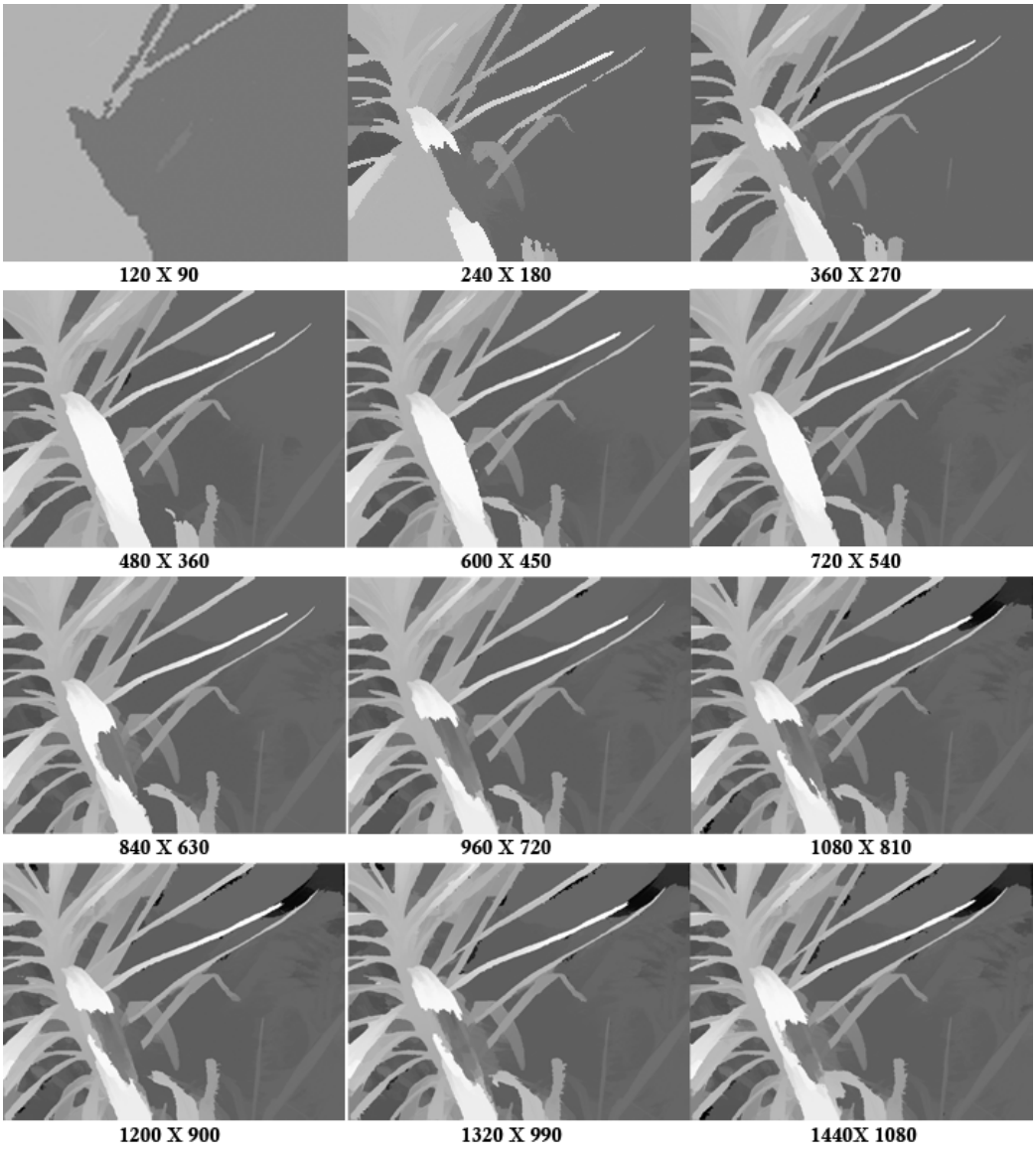


Figure 10: Results of disparity maps for the NB-Var configuration

3 Improvement

The term Φ_{sm} checks for every two pixels p and q if they are likely to have a smooth transition in the disparity map. As an improvement to the original algorithm, we are adding a 6th dimension to the feature vector $\mathbf{f}$ used in $w'_{p,q}$. In this sixth dimension we want to add a factor which shows how free a path there is between the two pixels, or in other words how many edges are encountered on the path between the pixels p and q.

We will first discuss where the idea comes from, and follow with the results of its implementation.

3.1 Idea

Whilst analyzing results of the original algorithm, we noticed that the depth map had some trouble in areas (see Figure 11). The disparity map is smoothed as if the pixels belong to the same object, even though there are clear edges to indicate otherwise. These edges are taken into account when the human eye looks at the image to determine which pixels belong together, but are not a part of $w'_{p,q}$. Therefor, the idea was formed to expand the 5 dimensional vector $\mathbf{f}$ with a 6th dimension: the edgeness e between the two pixels. If the two pixels can be connected without crossing many edges, it is most likely that there is a smooth transition between the two pixels' disparity. However, the heavier the edges we cross the more likely it is that there is not necessarily a smooth transition between the two pixels. We will define edge weight in section 3.3.

3.2 Getting the edges

To get the edges we need to determine the gradient at each pixel. The gradient is calculated by $\sqrt{\frac{\delta}{\delta x}^2 + \frac{\delta}{\delta y}^2}$. To get the derivatives $\frac{\delta}{\delta x}$ and $\frac{\delta}{\delta y}$ we use a Sobel kernel in the x and y direction.

3.2.1 Choosing the derivative input

The derivative of a color image can be defined in multiple ways. Whereas a gray scale image simply has one value to use as input for the derivative function, a color image can use i.a. hue, brightness, saturation and gray scale.

We compared the different derivatives, Figure 12 to 17, and found the gray scale and saturation as input gave very similar results, as can be seen

Figure 11: Problem areas

in close-up figures 15 and 16. Close comparison shows that the saturation channel seems to be at least as accurate or better at determining the gradient than the gray scale can.

Therefor, we choose not to use the gray scale image in our algorithm.

Since every other channel perform in some areas better at determining the gradient, we take the maximum of each gradient and store that as our final gradient value. To even the results of each channel, we normalize the contrast of the individual channel's gradients before taking the maximum, by dividing each value by the maximum value in the image.

For the result of the gradient, see Figure 18.

3.2.2 Sobel kernel size

The usual Sobel kernel is 1×3. We looked at different sizes of Sobel kernels, how they performed, and if there was a noticeable difference in runtime. The Sobel kernels we calculated can be found in table 19. Since Sobel kernels should be symmetrical, we tested kernels with size $1 \times k * 2 + 3$, where $0 \leq k \leq 5$. When a table has a "derivative number", we refer to the k of the size (e.g. derivative 4 has a dimension of $1 \times 4 * 2 + 3 \Rightarrow 1 \times 11$).

We compared each result shown in Figure 20 and although there are some slight differences in the size and value of the resulting gradients, all kernel sizes performed equal enough to not let their result be a factor in the choice of what to use in an algorithm. For a full comparison of run times, see section 4.1. We will use a kernel size of 1×13 in the rest of our experiments.

3.3 Assigning a value

To keep the idea of trying to maintain temporal stability by not having hard cuts in the algorithm, we want to add the edgeness as a factor to the weight formula $w_{p,q}$ in such a way that the more edge-pixels there are between the two pixels, the further apart the two pixels are in 6 dimensional space. We will first introduce the terms "path" and "thickness" before showing the adjusted formulas.

3.3.1 Paths

We will only look at the area between the two pixels to calculate the amount of edgeness between them. To determine this value we walk along the rasterized line between the two pixels, adding the change in value (delta) as we go along. The reason we are using a delta here, in stead of simply adding all values along the path, is because we do not want extra high values when walking

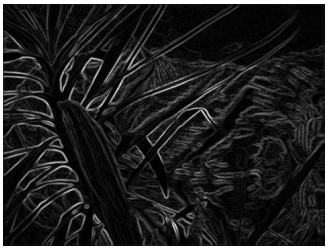


Figure 12: Gray scale derivative

Figure 13: Saturation derivative

Figure 14: Hue derivative

Figure 15: Gray scale derivative close-up

Figure 16: Saturation derivative close-up

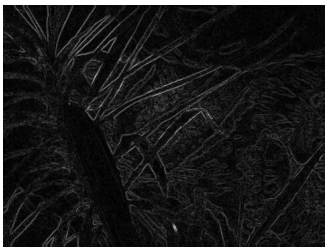
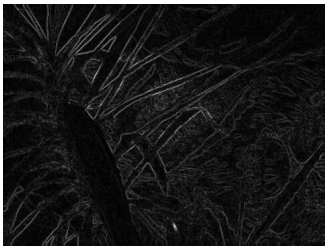


Figure 17: Brightness derivative

Figure 18: Max of all derivatives

	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
1×3						-0,5	0	0,5					
1×5					0,083333	-0,66667	5,95E-17	0,666667	-0,08333				
1×7				-0,01667	0,15	-0,75	1,06E-15	0,75	-0,15	0,016667			
1×9			0,003571	-0,0381	0,2	-0,8	4,49E-15	0,8	-0,2	0,038095	-0,00357		
1×11		-0,00079	0,009921	-0,05952	0,238095	-0,83333	-4,06E-14	0,833333	-0,2381	0,059524	-0,00992	0,000794	
1×13	0,000180	-0,0026	0,017857	-0,07937	0,267857	-0,85714	1,75E-13	0,857143	-0,26786	0,079365	-0,01786	0,002597	-0,00018

Figure 19: Sobel kernels



Sobel kernel size 1×3

Sobel kernel size 1×5

Sobel kernel size 1×7

Sobel kernel size 1×9

Sobel kernel size 1×11

Sobel kernel size 1×13

Figure 20: Gradient of the saturation channel with different Sobel kernel sizes

along an edge (e.g. as seen in Figure 21). The rasterization is according to Bressenham's line algorithm [1].

Let $\mathbf{L}$ be an in order array of the pixels along the rasterized line between p and q where $\mathbf{L}[0] = \mathbf{x}_p$ and $\mathbf{L}[\texttt{end}] = \mathbf{x}_q$. We define $e(\mathbf{x}_p, \mathbf{x}_q)$ as

$$e(\mathbf{x}_p, \mathbf{x}_q) = \sum_{i=0}^{end-1} |m(\mathbf{L}[i]) - m(\mathbf{L}[i+1])|$$

where $m(\mathbf{x})$ is the gradient in the pixel with coordinates $\mathbf{x}$.

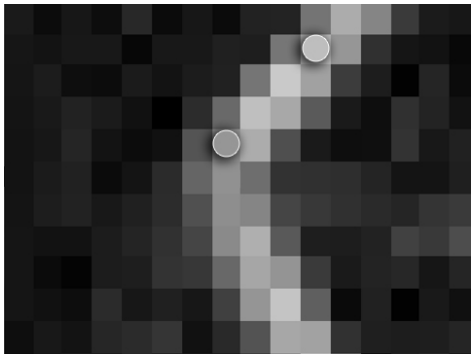


Figure 21: These two pixels can still belong to the same surface

Because we use the difference in gradient value along the path, this value can be positive or negative. We choose to use the absolute value of the delta, as to avoid averaging out to 0 when crossing a single line.

3.3.2 Multiple paths: thickness

When walking along a single path, the values can give us a distorted view of the likeliness of the two pixels belonging to the same surface. For example, in Figure 24 one can agree that the two pixels most likely do not belong to the same surface. The chance in Figure 23 is higher, which should be recognized by the algorithm.

The reason why the pixels in Figure 23 seem likely to belong to the same surface, is because there is a non-direct path visible between the two pixels. To be able to take those surrounding pixels into account, we introduce the thickness factor.

We call a thickness of 0 our basis, which is only the direct path between the pixels as previously described. For every thickness higher, we add a path to the left and right side of the original line, parallel, and on a distance of

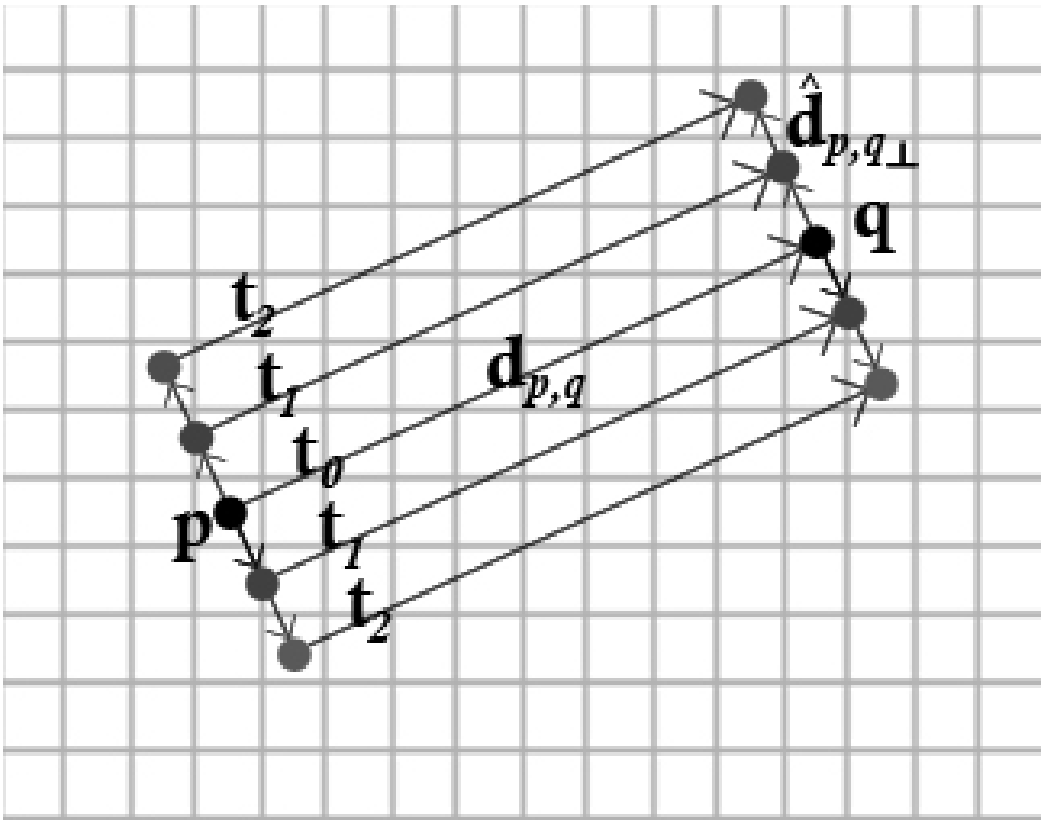


Figure 22: The 5 paths which are calculated at thickness 2

"thickness" pixels from the original line (see Figure 22). The final value is the average of all lines calculated. We use the average to achieve an effect of blurring the different lines. Using the minimum of these lines would gave the undesired effect of eroding the edges (see Figure 25 and 26).

Only lines which lie completely in the image are taken into account.

Assuming all lines lie completely within the image, the edgeness formula $e(\mathbf{x}_p, \mathbf{x}_q)$ is extended with thickness t as follows:

$$e(\mathbf{x}_p, \mathbf{x}_q, t) = \frac{1}{2t+1} \sum_{i=-t}^{t} e(\mathbf{x}_p + i * \hat{\mathbf{d}}_{p,q\perp}, \mathbf{x}_q + i * \hat{\mathbf{d}}_{p,q\perp})$$

where

$$\mathbf{d}_{p,q} = \mathbf{x}_q - \mathbf{x}_p$$

The vector $\hat{\mathbf{d}}_{p,q\perp}$ is the normalized vector perpendicular to the direction vector $\mathbf{d}_{p,q}$. To allow the discarding of lines which do not completely lie withing the image, we change the fraction to $\frac{1}{2t+1-l_{inv}}$, where l_{inv} is the amount of lines which are discarded.

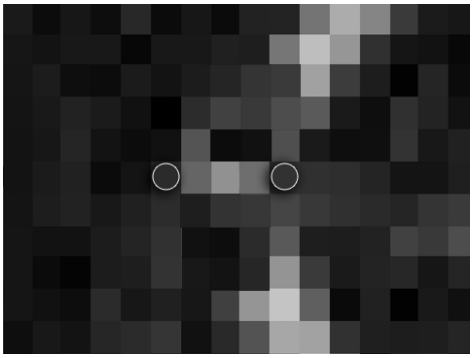


Figure 23: These two pixels probably belong to the same surface

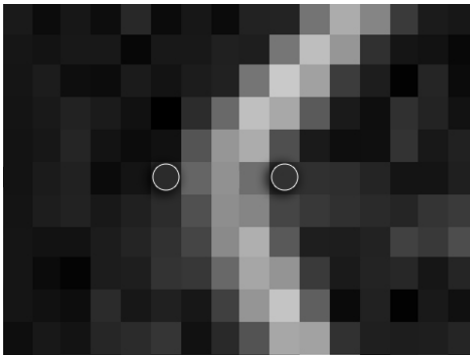


Figure 24: These two pixels probably do not belong to the same surface

3.3.3 Adjusted formulas

To integrate our edge feature in the weight formula $w'_{p,q}$, we change the formula as follows:

$$w'_{p,q} = f_x((\mathbf{x}_p - \mathbf{x}_q)^2) + f_c((\mathbf{c}_p - \mathbf{c}_q)^2) + f_e(e(\mathbf{x}_p, \mathbf{x}_q, t)^2)$$

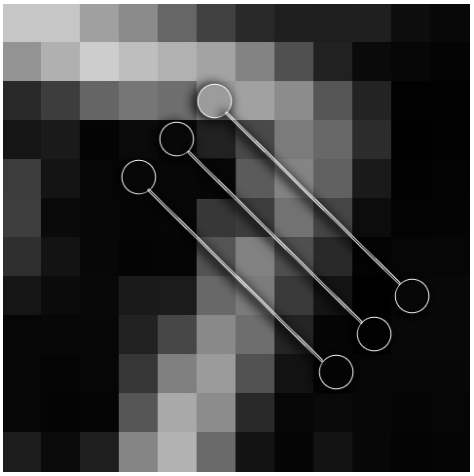


Figure 25: Thickness 1, original gradient values

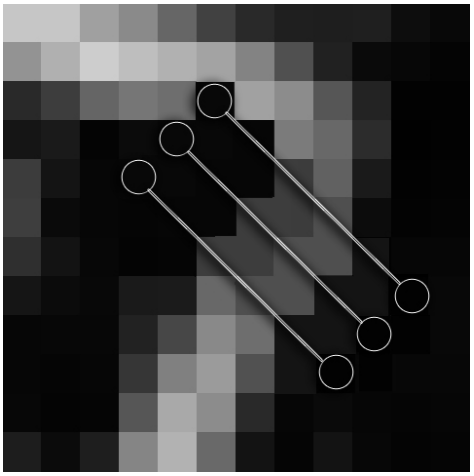


Figure 26: Thickness 1, effect of using minimum

where

$$f_e = \frac{1}{2\sigma_e^2}$$

with σ_e the bandwidth of the edgeness term.

4 Tests and results

4.1 Time measurements

All tests are run in a new test environment. The original program by Smith et al.[10] had multi-threading implemented. To keep the test runs as accurate as possible, all tests are run on a single core with a separate single threaded program. If we want to be able to compare the original algorithm to our expanded algorithm, we need a new baseline of our test setting. To obtain this baseline, we tested a single threaded program which calculated the distance of the pixels $w'_{p,q}$ via the NB-Var configuration. The results can be seen alongside the original programs results in Figure 27 where the new baseline is named "*C# Startup*".

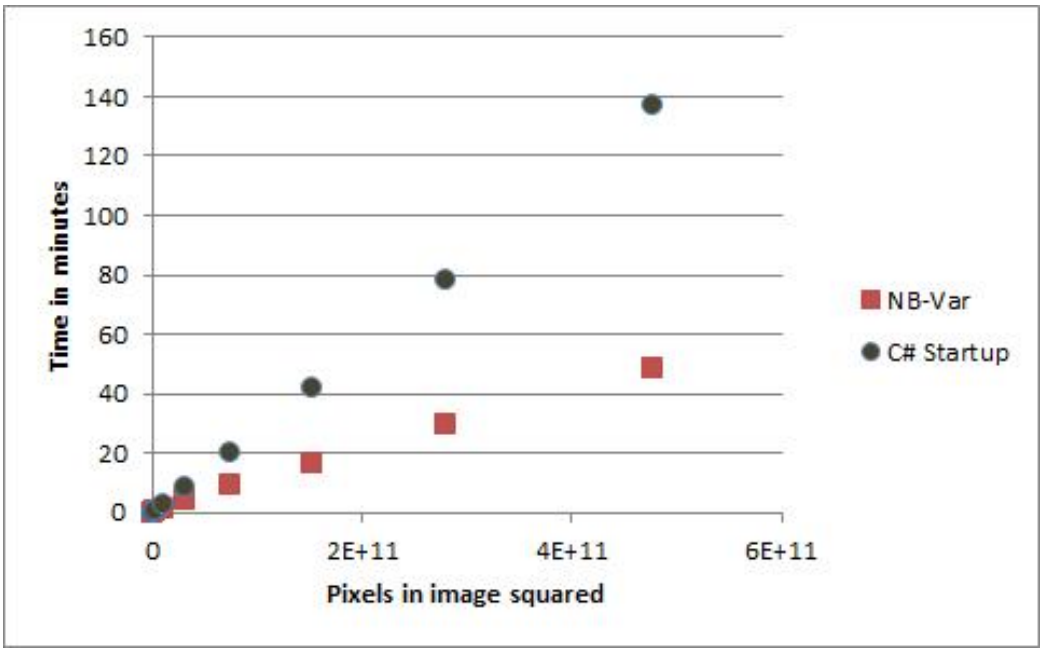


Figure 27: Average runtime of the NB-Var configuration and our test setting

To see how the different thickness and different kernel sizes compare in runtime, see Figure 28. Here we show the average runtime of the test on the y-axis, the thickness t on the x-axis, and the different kernel sizes as the different plotted lines. A close-up of the graph, in Figure 29 where only thickness 3 and 4 on the x-axis are shown, shows how kernel size 3 and 4 have a consistent lower runtime compared to kernel sizes 0, 1, 2 and 5. We cannot explain what causes this effect. We used kernel size 4 for all our further tests.

Figure 30 shows the expanded algorithm compared to the Baseline and

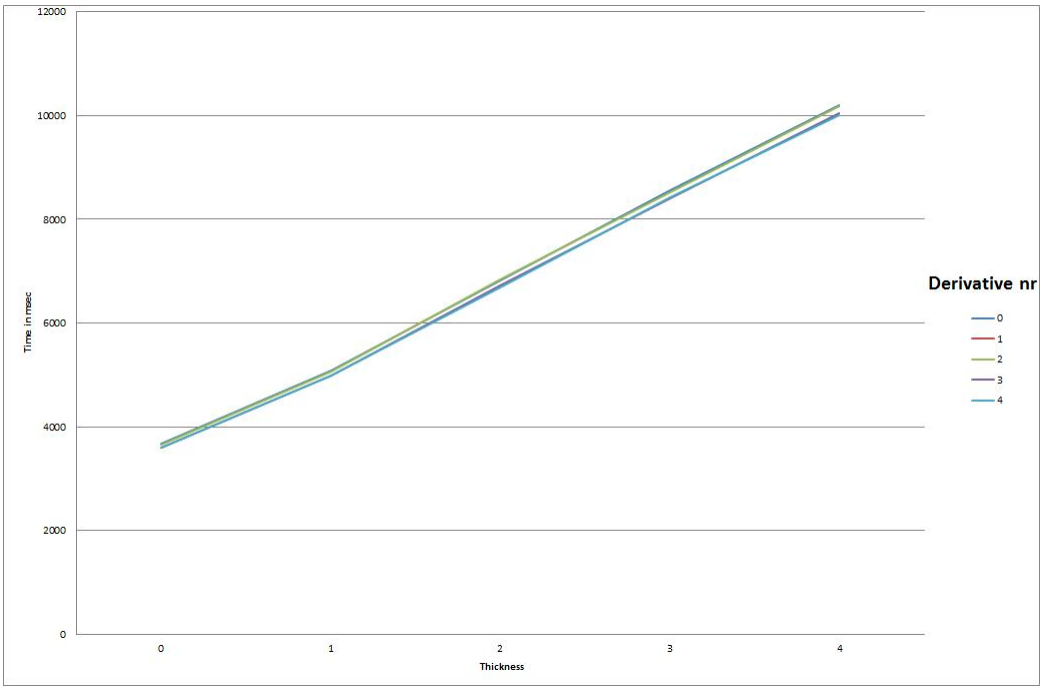


Figure 28: Average runtime when varying thickness and derivatives

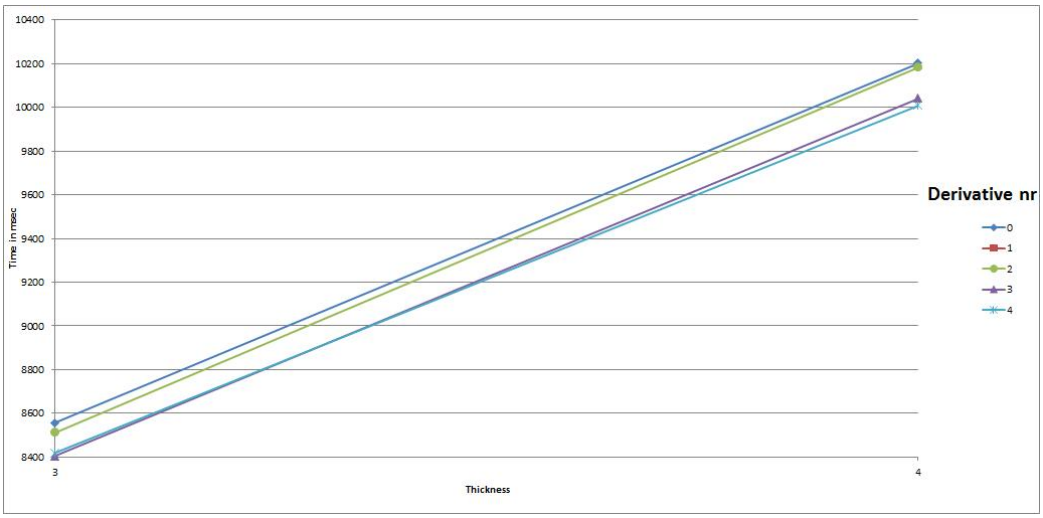


Figure 29: Average runtime when varying thickness and derivatives: closeup

the NB-Var configuration's measurements. We used a thickness of 0 and derivative kernel size 4. Although the algorithm has a higher runtime, with 539 minutes runtime on a 960×720 image compared to 137 minutes for the "*C# Startup*", it does appear to still behave linear in the amount of pixels. When we analyze our algorithm, we notice that we add a factor k to our algorithm where k is the maximum distance between p and q, making the calculation of the expanded $w'_{p,q}$ an $\mathcal{O}(n^2 * k)$ algorithm. If any size of image is allowed, we could make an image where $k = n$, with n the amount of pixels, making it an $\mathcal{O}(n^2 * n) \subseteq \Omega(n^3)$ algorithm. However in most use cases, the dimension of the picture will be somewhere between 1:1 and 16:9, in which case $\sqrt{2}\sqrt{n} \leq k \leq \sqrt{(\frac{16}{9})^2 + 1}\sqrt{n}$ making $k = c * \sqrt{n}$ with c a constant, resulting in an $\mathcal{O}(n^2\sqrt{n})$ algorithm.

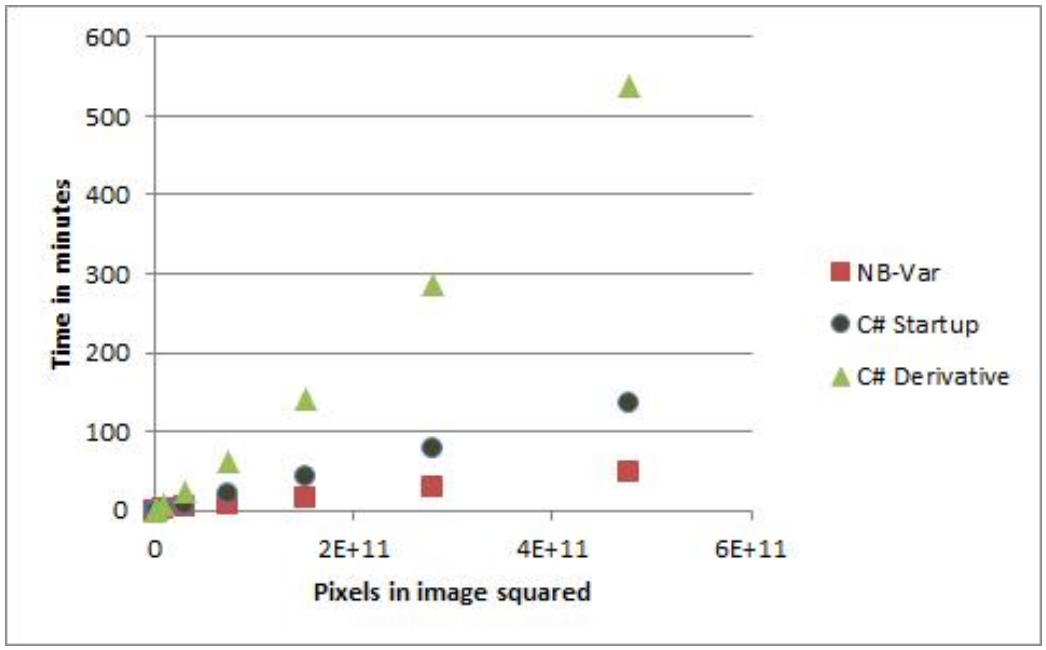


Figure 30: Average runtime of the NB-Var configuration, our test settings baseline, and the derivative enhanced algorithm

4.2 Visual Results

4.2.1 Thickness

Besides the difference in runtime, we also looked at the results for different thicknesses. Figure 31 to 35 show the visualized results of the gradient algorithm at position 1 of Figure 36. We visualized the values in such a way that

the lower the value the whiter the pixel. This means that white pixels are more likely to have a similar depth value. The visual results of thickness 0, Figure 31, shows that a single line to determine the edgeness of the surrounding pixels is prone to many false positives. The best results are obtained with thickness 1 and 2, showing a clear white triangle where the background is at position 1, whilst not blurring out as much as thickness 3 and 4 do. We choose thickness 2 for all further tests, because it shows a slightly less fickle behavior compared to a thickness of 1.

Figure 31: Visualized result of the derivative algorithm, thickness 0

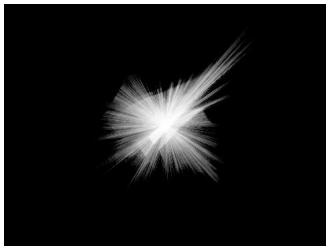


Figure 32: Visualized result of the derivative algorithm, thickness 1

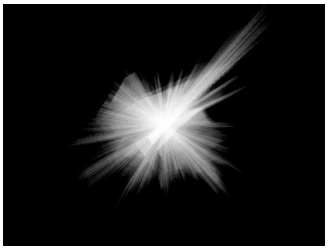


Figure 33: Visualized result of the derivative algorithm, thickness 2

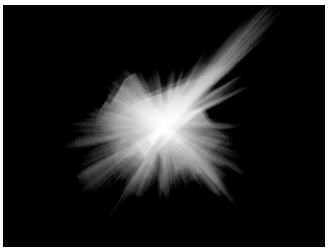


Figure 34: Visualized result of the derivative algorithm, thickness 3

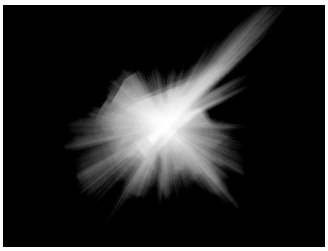


Figure 35: Visualized result of the derivative algorithm, thickness 4

4.2.2 Choosing f_e

The factor σ_e, the bandwidth of the gradient in the distance function, has to be chosen by hand. To show how different weights would affect the input for the rest of the algorithm, multiple factors have been examined. Because f_e is directly determined by σ_e, we list the different bandwidths in terms f_e, ranging from 1 to 30.

To illustrate the effect, 7 measuring points have been chosen in the image, see Figure 36. These points have been chosen to include problematic points

of the original algorithm (points 2, 3 and 6), possible problematic points of our addition(points 0, 4 and 5), and a point where the original algorithm performed correctly and we expect no negative effects from our addition (point 1).

The results can be seen for position 0 in Figures 36 to 45. All the results for all positions can be found in the Appendix, Figures 55 to 117.

Figure 36: Measuring points

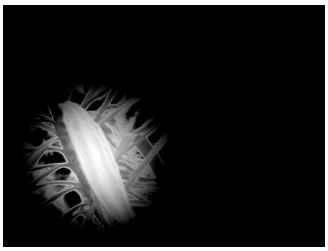


Figure 37: Original Algorithm, position 0

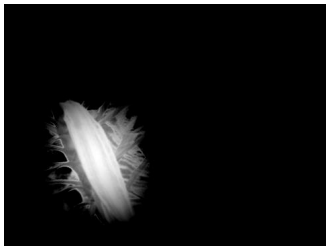


Figure 38: $f_e = 1$

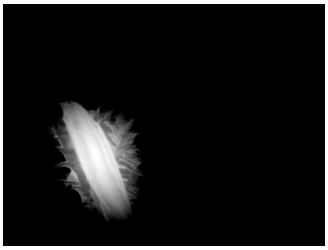


Figure 39: $f_e = 2$

Figure 40: $f_e = 5$

Figure 41: $f_e = 10$

Figure 42: $f_e = 15$

Figure 43: $f_e = 20$

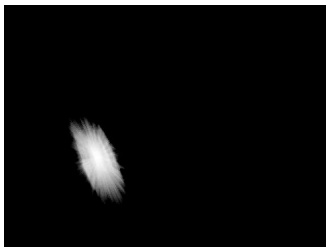


Figure 44: $f_e = 25$

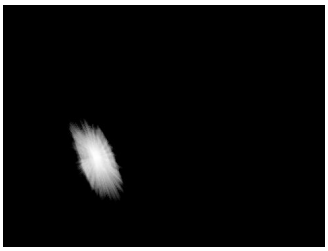


Figure 45: $f_e = 30$

Figure 46: Original Algorithm, position 6

Figure 47: $f_e = 1$

Figure 48: $f_e = 2$

Figure 49: $f_e = 5$

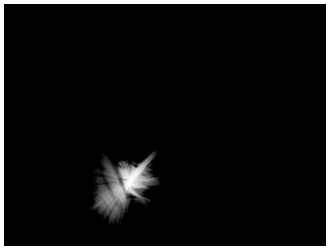


Figure 50: $f_e = 10$

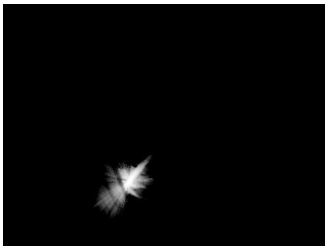


Figure 51: $f_e = 15$

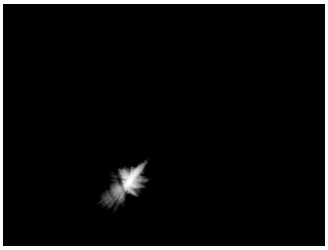


Figure 52: $f_e = 20$

Figure 53: $f_e = 25$

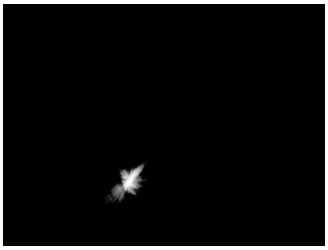


Figure 54: $f_e = 30$

5 Conclusion

The algorithm with the addition of the 6th dimension shows promising improvements when visualizing $w'_{p,q}$. The areas which were problem areas in the original algorithm's disparity map showed to have the problem existing in $w'_{p,q}$. We showed that those problem areas can have their false positives removed or reduced, whilst keeping the introduction of false negatives to a minimum.

The first thing to choose as a factor in the expansion is the thickness. As discussed in section 4.2.1, a minimum thickness of 1 is required for good results, and a thickness of 2 shows the best results of clear white pixels where we would identify the pixels as "belonging together" whilst at the same time not having as many false positives as higher thicknesses.

The factor f_e should be chosen in such a way that it removes false positives whilst keeping false negatives to a minimum. When choosing $f_e = 20$, the 7 different measuring points show this desired behavior. Notice how in 46 the center is surrounded by bright white pixels, which are all false positives resulting in the mislabeling of that pixel in the disparity map. When we look at 52, these false positives have been grayed out whilst leaving the leaf of the plant bright white.

The runtime of the algorithm is a concern. Although the original algorithm was in itself a slow but precise algorithm, a runtime of 539 minutes for a 960 image is too slow for many practical use.

However, the algorithm could be sped up by using memoization when calculating $e(\mathbf{x}_p, \mathbf{x}_q, t)$. At this time, every pixel recalculates the difference in gradient values along the rasterized line, even though if $\mathbf{x}_r \in \mathbf{L}$ then $e(\mathbf{x}_p, \mathbf{x}_q, t) = e(\mathbf{x}_p, \mathbf{x}_r, t) + e(\mathbf{x}_r, \mathbf{x}_q, t)$. We leave it up to future work to see what the actual speed up is when using the memoization.

6 Future work

First, we are interested in seeing how much the speed up is when using memoization, as discussed in the conclusion.

Secondly, we would be interesting to see if the disparity maps would improve as expected when integrating the new $w'_{p,q}$ into the algorithm.

Finally, we would like to test if we could find a more natural way to calculate the different thicknesses. At this time, the thickness is an easy thick line, but i.a. multiple arced lines from p to q might give a better result when looking at pixel which seem to belong together. Adjusting the thickness of the line along with the resolution of the image would also be advisable.

References

[1] Bressenhem, J.E. 1965. Algorithm for computer control of a digital plotter, *IBM Systems Journal* 4, 25 – 30.

[2] Klaus, A., Sormann, M., & Karner, K. 2006. Segmentbased stereo matching using belief propagation and a selfadapting dissimilarity measure. *Proceedings of the 18th International Conference on Pattern Recognition*, Volume 03, 15–18.

[3] Kolmogorov, V., & Zabih, R. 2002. Multi-camera scene reconstruction via graph cuts, *European Conference on Computer Vision 2002*, 82–96.

[4] Liu, C., Huang, T., Chang, M., Lee, K., Liang, C. & Chuang, Y. 2011. 3D cinematography principles and their applications to stereoscopic media processing. *Proceedings of the 19th ACM international conference on Multimedia* , 253–262.

[5] Lo, W., van Baar, J., Knaus, C., Zwicker, M., & Gross, M. 2010. Stereoscopic 3D Copy & Paste. *ACM Trans. Graph. 29*, 6, Article 147.

[6] Luo, S., Shen, I., Chen, B., Cheng, W, & Chuang, Y. 2012. Perspective-aware Warping for Seamless Stereoscopic Image Cloning, *ACM Trans. Graph. 31*, 6, Article 182.

[7] Scharstein, D., & Szeliski, R. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, Volume 47 Issue 1-3, 7–42.

[8] Scharstein, D., & Szeliski, R. 2013. Middlebury Stereo Evaluation - Version 2. Retrieved 27-09-2013 from *http://vision.middlebury.edu/stereo/eval/*.

[9] Smith, B.M., Zhang, L., & Hailin, J. 2009. Stereo Matching with Nonparametric Smoothness Priors in Feature Space. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2009*, 458-492.

[10] Smith, B.M., Zhang, L., & Hailin, J. 2009. Stereo Matching with Nonparametric Smoothness Priors in Feature Space - CVPR 2009. Retrieved 27-09-2013 from *http://pages.cs.wisc.edu/ lizhang/projects/mvstereo/cvpr2009/*.

[11] Tornow, M., Grasshoff, M., Nguyen, N., Al-Hamadi, A., & Michaelis, B. 2012. Fast Computation of Dense and Reliable Depth Maps from

Stereo Images. *Machine Vision - Applications and Systems*, Available from: *http://www.intechopen.com*

[12] Woodford, O.J., Torr, P.H.S., Reid, I.D., & Fitzgibbon, A.W. 2008. Global stereo reconstruction under second order smoothness priors, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2008*, 1–8.

[13] Zhu, S., Zhang, L. & Hailin, J. 2012.A Locally Linear Regression Model for Boundary Preserving Regularization in Stereo Matching. Retrieved 06-07-2014 from *http://pages.cs.wisc.edu/ lizhang/projects/llr-stereo/*.

7 Appendix

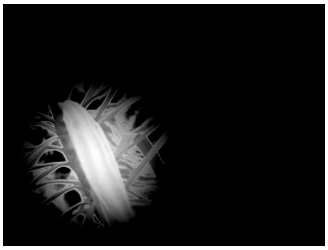


Figure 55: Original Algorithm, position 0

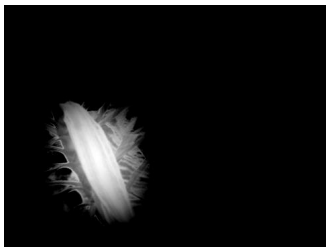


Figure 56: $f_e = 1$

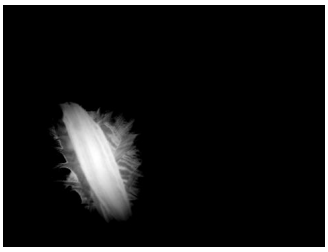


Figure 57: $f_e = 2$

Figure 58: $f_e = 5$

Figure 59: $f_e = 10$

Figure 60: $f_e = 15$

Figure 61: $f_e = 20$

Figure 62: $f_e = 25$

Figure 63: $f_e = 30$

Figure 64: Original Algorithm, position 1

Figure 65: $f_e = 1$

Figure 66: $f_e = 2$

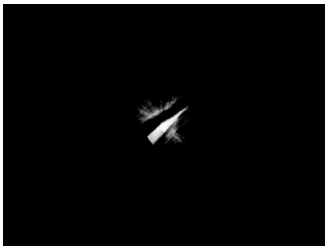


Figure 67: $f_e = 5$

Figure 68: $f_e = 10$

Figure 69: $f_e = 15$

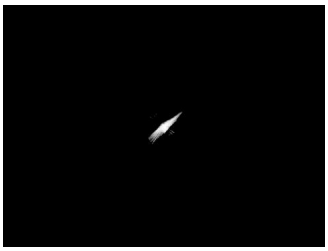


Figure 70: $f_e = 20$

Figure 71: $f_e = 25$

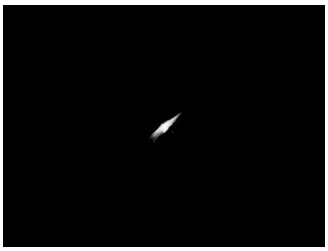


Figure 72: $f_e = 30$

Figure 73: Original Algorithm, position 2

Figure 74: $f_e = 1$

Figure 75: $f_e = 2$

Figure 76: $f_e = 5$

Figure 77: $f_e = 10$

Figure 78: $f_e = 15$

Figure 79: $f_e = 20$

Figure 80: $f_e = 25$

Figure 81: $f_e = 30$

Figure 82: Original Algorithm, position 3

Figure 83: $f_e = 1$

Figure 84: $f_e = 2$

Figure 85: $f_e = 5$

Figure 86: $f_e = 10$

Figure 87: $f_e = 15$

Figure 88: $f_e = 20$

Figure 89: $f_e = 25$

Figure 90: $f_e = 30$

Figure 91: Original Algorithm, position 4

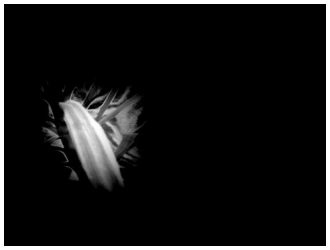


Figure 92: $f_e = 1$

Figure 93: $f_e = 2$

Figure 94: $f_e = 5$

Figure 95: $f_e = 10$

Figure 96: $f_e = 15$

Figure 97: $f_e = 20$

Figure 98: $f_e = 25$

Figure 99: $f_e = 30$

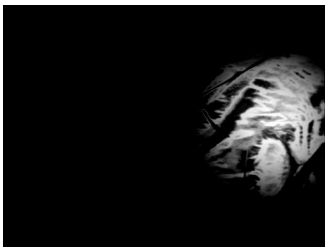


Figure 100: Original Algorithm, position 5

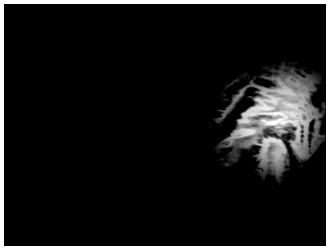


Figure 101: $f_e = 1$

Figure 102: $f_e = 2$

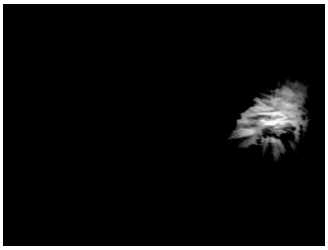


Figure 103: $f_e = 5$

Figure 104: $f_e = 10$

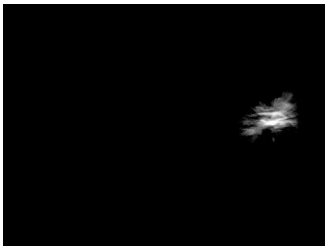


Figure 105: $f_e = 15$

Figure 106: $f_e = 20$

Figure 107: $f_e = 25$

Figure 108: $f_e = 30$

Figure 109: Original Algorithm, position 6

Figure 110: $f_e = 1$

Figure 111: $f_e = 2$

Figure 112: $f_e = 5$

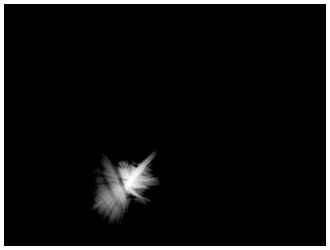


Figure 113: $f_e = 10$

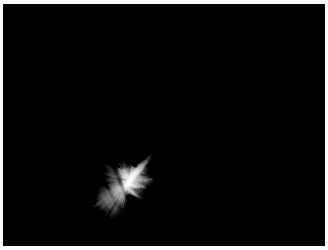


Figure 114: $f_e = 15$

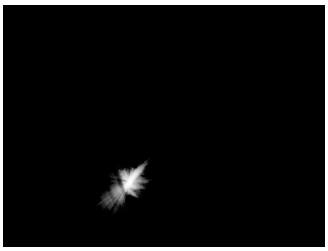


Figure 115: $f_e = 20$

Figure 116: $f_e = 25$

Figure 117: $f_e = 30$