3. Snake or Parametric Active Contour

Many of the original or classical image segmentation techniques like thresholding or edge detection techniques are very susceptible to noise. Another issue that causes problems are gaps in the boundary. As seen in chapter two watershed methods also perform poorly under these conditions. The snake or parametric active contour was developed to handle these problems. They perform well in spite of noise and boundary gaps.

This chapter has been done in cooperation with Aurelie Bugeau an exchange student from the 'Higher National Engineering School of Electronics, Computer Science and Telecommunications of Bordeaux' in France, who investigated and implemented certain enhancements on the snake method.

First we treat the theory of snakes, then the implementation of the snake is given. Some improvements that have been developed are discussed in the following sections, as applying the snake on multiple scales and with the gradient vector flow force. The balloon snake and a semi-automatic initialization developed by Aurelie Bugeau are discussed next. Then the robustness of the snake method is tested by applying it to some artificial and real images. Finally a discussion on this method is given and some suggestions for future enhancements.

3.1 Theory

In 1988 M.Kass, A.Witkin and D.Terzopoulos [13] proposed snakes as a way to detect edges and to segment images. The basic idea is that a parameterized curve is drawn outside or inside the object that is to be segmented. This curve or contour is attracted to or pushed away from certain features of the object in the image. This can be compared with an elastic band that is stretched out and tries to contract back into its original shape against a force. The elastic represents the contour and the force is derived from the image.

The total energy of an active contour with parametric representation $V(s) = (X(s), Y(s))$ can be written as follows:

$$E_{\text{total}} = \int_0^1 E(V(s)) \, ds = \int_0^1 E_{\text{int}}(V(s)) + E_{\text{ext}}(V(s)) \, ds$$

In this equation $s$ is the parameterization, $s \in [0,1]$, $E_{\text{total}}$ is the total energy, $E_{\text{int}}$ is the internal energy given in equation 3.2 and $E_{\text{ext}}$ is the external energy given in equation 3.3. The contour stops moving when the internal and external energies are at equilibrium. In other words when the minimum energy difference is obtained [2, 24]. Equation 3.1 states that the total energy is equal to the sum of internal and external energies for each pixel on the entire contour. This equation should be minimal on the boundaries of the object to be segmented. The internal energy is given in the following equation:

$$E_{\text{int}} = E_{\text{cont}} + E_{\text{curv}} = \alpha(s) \left| \frac{\partial Y}{\partial s} \right|^2 + \beta(s) \left| \frac{\partial X}{\partial s} \right|^2$$

In this equation $E_{\text{cont}}$ is the continuity energy which ensures that the parameterization points remain equidistant from each other. $E_{\text{curv}}$ is the curvature energy which maintains the rigidity of the snake. The higher the rigidity, the smoother the curve. $\alpha$ and $\beta$ are constants which can be used to control the importance of the two energies respectively. The external energy is usually taken to be the gradient magnitude of the image.

$$E_{\text{ext}} = -\gamma \left| \nabla G_\sigma(x, y) \ast I(x, y) \right|^2$$

In this equation $G_\sigma(x,y)$ is a Gaussian kernel with a scale $\sigma$ which is convolved with the image $I(x,y)$. As a result the image is blurred, $\gamma$ is a constant which can be used to control the importance of the external energy.
Another way of representing the snake that is often used in literature is by considering the contour to move under the influence of forces [24].

\[ \gamma \frac{\partial V}{\partial t} = F_{\text{ext}}(V) + F_{\text{int}}(V) \]  \hspace{1cm} (3.4)

In this equation, \( F_{\text{int}} \) represents the internal forces of the snake and \( F_{\text{ext}} \) the external forces working on the snake. These are given by the following equations.

\[
F_{\text{int}}(V) = \frac{\partial}{\partial s} \alpha(s) \frac{\partial V}{\partial s} - \frac{\partial^2}{\partial s^2} \beta(s) \frac{\partial^2 V}{\partial s^2} \]  \hspace{1cm} (3.5)

\[
F_{\text{ext}}(V) = -\gamma \nabla G_{s}(x, y) \ast I(x, y) \]  \hspace{1cm} (3.6)

In this representation, additional forces can simply be inserted into equation 3.4.

Several different external forces have been developed over the years. Many of these additional forces have been developed to combat limitations of the snake that have been observed. One of the limitations is that the snake must be initialized close to the edge of the object that is to be segmented. Several methods have been developed to extend the attraction range of the snake.

The attraction range of the snake can be improved by a multiscale Gaussian potential force proposed by Terzopoulos, Witkin and Kass [26]. By applying the basic snake method discussed above on successively smaller \( \sigma \) for the Gaussian kernel in equation 3.3 or 3.6 the initial contour could be placed further away.

By using the Euclidean distance of the pixels to the closest boundary point as the external energy another solution can be found to increase the attraction range of the snake, as was proposed by Cohen and Cohen [5].

Another solution that addresses this problem is the addition of a pressure force as proposed by Cohen [6]. In this method a constant force is added to equation 3.4 that works in the normal direction of the contour.

When the force is negative the contour moves inward, when the force is positive it moves outward. This method improves the attraction range of the snake, but also helps the snake enter concavities. In section 3.5 the pressure force is investigated further.

A problem with the pressure force is that it can cause the snake to cross weak edges on which it actually should remain. A nice solution to this problem is given by Jim Ivins and John Porril [12]. They propose to use statistical means to drive the pressure force. The mean intensity and its standard deviation of the object is determined. The same is determined for the area outside the object. The pressure force is made dependent on the intensity of the location the contour point is currently on. Is the intensity in the outside range, then the pressure force points inward, should it be in the inside range, then the pressure force points outward.

Another problem that can be observed in the basic snake is that it has trouble entering concavities. The pressure force as mentioned above alliviates this problem somewhat.

A better method is given by Xu [30] in the form of gradient vector flow. In this method the gradient of the image is diffused over the image. As a result the snake enters concavities much better. This method is discussed in more detail in section 3.4.

A major problem with the snake method is that it is very hard to split one contour in multiple contours or merge multiple contours into one. To be able to track the contour points of the snake when a merge or split occurs, McInerney and Terzopoulos [18] developed a method that uses a superimposed simplicial grid and reparameterization of the contour.

To handle exactly this problem the levelset method was applied on the active contour principle by Malladi [16] and Casselles [4]. In this method the contour is embedded in a higher dimensional function and this
ensures that merging and splitting can occur naturally. This method is discussed in more detail in the next chapter.

Completely different approaches of the snake have also been investigated by for instance Perrin and Smith [19]. In their method instead of using the derivatives of the contour to represent the elastic force, they use the angles between the lines in the contour points as a measure of the tension.

The basic snake is implemented in the next section.

### 3.2 Implementation

First an image is created on which the snake will be demonstrated. The image is 256 by 256 and has a circle of intensity one and a radius of 50 pixels on a black background. The snake operates on the gradient magnitude image, which is calculated using the gD function from the Front-End Vision book [10].

```mathematica
<< FrontEndVision`FEV`
image = Table[If[(x)^2 + (y - 30)^2 < 2500, 1, 0], {y, -127.5, 127.5}, {x, -127.5, 127.5}];
imgrad = Sqrt[gD[image, 1, 0, 3]^2 + gD[image, 0, 1, 3]^2];
DisplayTogetherArray[{ListDensityPlot[image], ListDensityPlot[imgrad]}];
```

![Image of test image and its gradient magnitude](image)

**Figure 3.1:** The test image and its gradient magnitude, with $\sigma = 3$.

The first contour is user defined. This is done by simply giving some pixels through which the first contour is drawn. These pixels should be chosen not too far away from the edge of the object that is to be segmented.

```mathematica
inputpixels = {{70.0142, 161.932}, {72.3277, 177.246}, {78.1667, 191.347},
{88.743, 205.449}, {105.268, 216.025}, {128.734, 218.339}, {153.412, 212.5},
{173.353, 197.186}, {183.929, 176.034}, {183.929, 147.83},
{172.251, 125.466}, {155.726, 112.576}, {134.573, 104.314},
{106.37, 110.263}, {85.2175, 121.941}, {72.3277, 143.093}};
```
 Every contour point is now considered individually. The neighborhood of a contour point determines which way the point will move and thus which way the snake moves. This is depicted in figure \(\text{3.3}\).

The total energy is calculated for every pixel in the neighborhood of the contour point: so the energy of the snake is calculated considering the contour point to be on the pixels in its neighborhood. Then the position in which the total energy is the smallest is chosen as the next contour point. In this way the snake moves toward the local minimal energy configuration.

Now the internal energies are calculated. There are two internal energies. The first is the continuity energy. This can be thought of as representing the elastic property of the contour. It is calculated by looking at the distance between two contour points compared to the average distance between all contour points. The second energy is the curvature energy. This can be thought of as representing the rigid property of the contour. The more rigid the contour is the more the contour will be a straight line between three contour points. It is calculated by looking at the distance between a point and the point before and after it. This can be seen as the second derivative of the contour.

Both the internal energies of the snake are dependent on the distance between consecutive points. Thus a function is necessary that calculates the difference between two points.
\[ \text{distance}[v1, v2] := \sqrt{\text{Plus} @@ (v1 - v2)^2}; \]

In which \( v1 \) and \( v2 \) are the two points.

The contour energy is calculated by first determining the mean distance of the contour points and for each pixel in the neighborhood of a contour point determine the difference with this mean. The following function calculates the contour energy.

\[ \text{compecont}[d, vj, \text{vimin1}] := d - \text{distance}[vj, \text{vimin1}]; \]

In which \( d \) is the mean distance of the contour points, \( \text{vimin1} \) is the location of the previous contour point and \( vj \) is the location of the current contour point.

The average distance between the contour points is calculated for the contour.

\[ d = N[\text{Plus} @@ \text{MapThread}[\text{distance}, \{\text{inputpixels}, \text{RotateLeft[}\text{inputpixels}]\}]] / \text{Length[\text{inputpixels}]}; \]

The continuity energy is calculated for the neighborhood of each contour point according to the \text{compecont} function above and normalized, in such a way that the continuity energy is between 0 and 1 for each neighborhood.

\[ \text{econt} = \text{MapThread}[\text{Table}[\text{compecont}[d, \#1 + \{1, j\}, \#2], \{i, -1, 1\}, \{j, -1, 1\}] \&, \{\text{Round[\text{inputpixels}]}, \text{Round[\text{RotateRight[}\text{inputpixels}]\}]\}]; \]

\[ \text{econt} = \frac{\# - \text{Min[\#]} \& / \text{Abs[\text{econt}]}}{\text{Max[\#]} - \text{Min[\#]}} \]

The following function calculates the curvature energy, as can be seen this is a discrete version of calculating the second derivative.

\[ \text{compecurv[viplus1, vj, \text{vimin1}] := Plus @@ ((viplus1 - 2 vj + \text{vimin1})^2)} \]

In which \( \text{viplus1}, vj \) and \( \text{vimin1} \) are the locations of the previous, current and next contour points respectively.

The curvature energy is calculated according to the \text{compecurv} function above for the neighborhood of each contour point and normalized, in such a way that the curvature energy is between 0 and 1 for each neighborhood.

\[ \text{ecurv} = \text{MapThread}[\text{Table}[\text{compecurv}[\#1, \#2 + \{1, j\}, \#3], \{i, -1, 1\}, \{j, -1, 1\}] \&, \{\text{Round[\text{RotateLeft[}\text{inputpixels}]\}], \text{Round[\text{inputpixels}]}, \text{Round[\text{RotateRight[}\text{inputpixels}]\}]\}]; \]

\[ \text{ecurv} = \frac{\# - \text{Min[\#]} \& / \text{Abs[\text{ecurv}]}}{\text{Max[\#]} - \text{Min[\#]}} \]

Now the gradient around the contour points is taken into account, this determines the external energy. The maximum and minimum of the neighborhoods of the contour pixels is calculated as follows.

\[ \text{blocks} = \text{Take[imgrad, \{First[\#] - 1, First[\#] + 1\}, \{Last[\#] - 1, Last[\#] + 1\}] \& / \text{Round[\text{inputpixels}]}; \text{gmin} = \text{Min} / \text{blocks}; \text{gmax} = \text{Max} / \text{blocks}; \]

The gradient of the neighborhoods is now normalized in the following manner. First a cutoff is defined under this value the pixel will be reset to the maximum value minus this threshold value.

\[ \text{gthreshold} = (\text{Max[imgrad]} - \text{Min[imgrad]}) / 1000; \]
First it has to be tested that there will be no division by zero. Then the neighborhoods with the gradient magnitude around the contour points are normalized between -1 and 0.

\[
g_{\text{test}} = g_{\text{max}} - g_{\text{min}}; \quad g_{\text{min}} = \text{MapThread}[\text{If}[#1 < \text{gthreshold}, #2 - \text{gthreshold}, #3] &, \{g_{\text{test}}, g_{\text{max}}, g_{\text{min}}\}];
\]

\[
g = -\frac{\text{blocks} - g_{\text{max}}}{g_{\text{max}} - g_{\text{min}}};
\]

This gives the external energy which is determined from the gradient image. Every energy is normalized between 1 and 0 or -1 and 0 so that the energies can be compared to each other. The total energy is now calculated according to equation 3.1 with weight factors for the energies. By tuning these weight factors the significance of the continuity, curvature or image energy can be adjusted.

\(\alpha\) is the weight factor for the elasticity, the higher this constant the more the contour points will be drawn towards each other.

\(\beta\) is the weight factor for the curvature, the higher this constant the more the contour points will behave as if they are part of rigid rods.

\(\gamma\) is the weight factor for the image energy, the higher this constant the more the contour points will be attracted by the image features.

\[
\alpha = 1; \quad \gamma = 1; \quad \beta = 0.5;
\]

\[
\text{totalEnergy} = \alpha \text{econt} + \beta \text{ecurv} + \gamma g;
\]

Now the pixels in each neighborhood with the minimum energy are chosen as the new contour points.

\[
\text{incr} = (-2, -2) + \text{Flatten}[	ext{First}[	ext{Position}[#, \text{Min}[#]], 1]) & /@\text{Abs}[\text{totalEnergy}];
\]

\text{nrmoved} gives an indication of how much the contour moved. If this is very low the minimum energy is probably reached.

\[
\text{nrmoved} = \text{Plus} @@\text{Plus} @@\text{Abs}[\text{incr}];
\]

The contour points are updated.
newinputpixels = inputpixels + incr;
inputp = Replace[#, {a_, b_} :> {b, a}] & /@ inputpixels;
newinputp = Replace[#, {a_, b_} :> {b, a}] & /@ newinputpixels;
gr = {Red, Thickness[0.01], Spline[Append[inputp, First[inputp]], Cubic]};
gr2 =
{Red, Thickness[0.01], Spline[Append[newinputp, First[newinputp]], Cubic]};
ListDensityPlot[image, Epilog -> {Green, PointSize[.02], Point /@ inputp, 
gr, Pink, Point /@ newinputp}, Mesh -> False, PlotRange -> All];
inputpixels = newinputpixels;

Figure 3.4: Contour points updated, in which the green dots are the previous contour points and the pink dots the new ones.

This is repeated until the number of contour points that moved is very low; in the appendix the continuous implementation is given. This is applied on the test image created before as an illustration of how the snake evolves the contour. For this purpose it is applied on a simple test image.

Figure 3.5: Final result of snake algorithm on circle.

To get a better feeling of the influence of the parameters of the snake, $\alpha$ and $\beta$ are now varied. First $\alpha$ is varied from 1, 0.5 to 0, while $\beta$ is kept 1. The result is given in figure 3.6.
As can be seen in figure 3.6, by decreasing $\alpha$ the distance between the contour points is less and less equal. When $\alpha$ is equal to zero the contour points leave big gaps between each other. Next the $\beta$ is varied from 1, 0.5 to 0, while $\alpha$ is kept 1. The result is given in figure 3.7.

As can be seen in figure 3.7, by decreasing $\beta$ the contour becomes less rigid and sharper angles can be formed between contour points. This can give dramatic results as in the right image. When $\beta$ is too big though as in the left image, the contour points can't handle corners well. This can especially be seen by comparing the left image with the middle image, where the corners are taken much better. Therefore care must be taken as to how the parameters are chosen.

### 3.3 Multi-Scale

A drawback of the snake is that the initialization of the first contour must be done very close to the edge of the object that is to be detected. One way to deal with this is let the snake perform on a highly blurred version of the gradient image first. When the snake has reached local equilibrium the scale of the blurring is reduced and the snake will adjust itself to the more detailed gradient image. This is done continuously until the scale is small, say one pixel. By then the snake should have adapted to the details of the image nicely. In this way the snake first adapts to large structures in the image and more and more to the more detailed structures of the image. This is also called edge focusing, as the edges of the structures become clearer with each time the scale is decreased [10].

Now the initialization of the first contour does not have to be done as close to the edge of the object. The largest scale of the blurring determines how far away the input pixels can be placed. The final result of each scale becomes the initial contour for the next scale. This has been done for the test image given before.
An alternative snake method is the gradient vector flow snake that was developed by Chenyang Xu and Jerry L. Prince [30]. It handles the small attraction range of the snake as well. The gradient vector field is used as another force derived from the image that works on the snake, but as opposed to the previous force derived from the image, namely the gradient magnitude, it is not a scalar, but a vector. By applying the gradient vector field the snake not only has a larger attraction range, but enters concavities in the border more easily.

The gradient of an image can be depicted as a vector field in which the derivative in the x direction and the derivative in the y direction determine the direction and size of the vector. The gradient vector flow field can now be visualized as the gradient of an image diffused out over the image. Near the edge the gradient vector flow field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.

The gradient vector flow field is defined as the vector field $\vec{v}(x, y) = (u(x, y), v(x, y))$ that minimizes the following energy function.

$$\varepsilon = \int \int_\Omega \left( \mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) + \left\| \nabla f - \nabla \vec{v} \right\|^2 \right) \, dx \, dy \quad (3.7)$$

In which $\mu$ is a constant, $u_x$, $u_y$, $v_x$, and $v_y$, are the derivatives of $u$ and $v$ in the x and y direction respectively and $f$ is the image gradient magnitude.

When $\nabla f$ is small, which occurs in homogeneous areas, the energy is a result of the partial derivatives of the vector field. When the energy is minimized the derivatives of the vector field become smaller, thus the vector

```
DisplayTogetherArray[
ListDensityPlot[\[Sqrt]gD[image, 1, 0, \sigma]^2 + gD[image, 0, 1, \sigma]^2] \& /\@ 
{20, 10, 5, 1}];
```

Figure 3.6: Blurred gradient magnitude image with scale 20, 10, 5 and 1 from left to right.

The result is given in figure 3.7 on which the final contour is drawn on the gradient image to give an insight in how the contour is guided to the edge.

```
Figure 3.7: Final result of multiscale snake, whereby the final contour is drawn on the gradient magnitude image.
```

### 3.4 Gradient Vector Flow

An alternative snake method is the gradient vector flow snake that was developed by Chenyang Xu and Jerry L. Prince [30]. It handles the small attraction range of the snake as well. The gradient vector field is used as another force derived from the image that works on the snake, but as opposed to the previous force derived from the image, namely the gradient magnitude, it is not a scalar, but a vector. By applying the gradient vector field the snake not only has a larger attraction range, but enters concavities in the border more easily.

The gradient of an image can be depicted as a vector field in which the derivative in the x direction and the derivative in the y direction determine the direction and size of the vector. The gradient vector flow field can now be visualized as the gradient of an image diffused out over the image. Near the edge the gradient vector flow field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.

The gradient vector flow field is defined as the vector field $\vec{v}(x, y) = (u(x, y), v(x, y))$ that minimizes the following energy function.

$$\varepsilon = \int \int_\Omega \left( \mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) + \left\| \nabla f - \nabla \vec{v} \right\|^2 \right) \, dx \, dy \quad (3.7)$$

In which $\mu$ is a constant, $u_x$, $u_y$, $v_x$, and $v_y$, are the derivatives of $u$ and $v$ in the x and y direction respectively and $f$ is the image gradient magnitude.

When $\nabla f$ is small, which occurs in homogeneous areas, the energy is a result of the partial derivatives of the vector field. When the energy is minimized the derivatives of the vector field become smaller, thus the vector field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.

```
DisplayTogetherArray[
ListDensityPlot[\[Sqrt]gD[image, 1, 0, \sigma]^2 + gD[image, 0, 1, \sigma]^2] \& /\@ 
{20, 10, 5, 1}];
```

Figure 3.6: Blurred gradient magnitude image with scale 20, 10, 5 and 1 from left to right.

The result is given in figure 3.7 on which the final contour is drawn on the gradient image to give an insight in how the contour is guided to the edge.

```
Figure 3.7: Final result of multiscale snake, whereby the final contour is drawn on the gradient magnitude image.
```

### 3.4 Gradient Vector Flow

An alternative snake method is the gradient vector flow snake that was developed by Chenyang Xu and Jerry L. Prince [30]. It handles the small attraction range of the snake as well. The gradient vector field is used as another force derived from the image that works on the snake, but as opposed to the previous force derived from the image, namely the gradient magnitude, it is not a scalar, but a vector. By applying the gradient vector field the snake not only has a larger attraction range, but enters concavities in the border more easily.

The gradient of an image can be depicted as a vector field in which the derivative in the x direction and the derivative in the y direction determine the direction and size of the vector. The gradient vector flow field can now be visualized as the gradient of an image diffused out over the image. Near the edge the gradient vector flow field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.

The gradient vector flow field is defined as the vector field $\vec{v}(x, y) = (u(x, y), v(x, y))$ that minimizes the following energy function.

$$\varepsilon = \int \int_\Omega \left( \mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) + \left\| \nabla f - \nabla \vec{v} \right\|^2 \right) \, dx \, dy \quad (3.7)$$

In which $\mu$ is a constant, $u_x$, $u_y$, $v_x$, and $v_y$, are the derivatives of $u$ and $v$ in the x and y direction respectively and $f$ is the image gradient magnitude.

When $\nabla f$ is small, which occurs in homogeneous areas, the energy is a result of the partial derivatives of the vector field. When the energy is minimized the derivatives of the vector field become smaller, thus the vector field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.

```
DisplayTogetherArray[
ListDensityPlot[\[Sqrt]gD[image, 1, 0, \sigma]^2 + gD[image, 0, 1, \sigma]^2] \& /\@ 
{20, 10, 5, 1}];
```

Figure 3.6: Blurred gradient magnitude image with scale 20, 10, 5 and 1 from left to right.

The result is given in figure 3.7 on which the final contour is drawn on the gradient image to give an insight in how the contour is guided to the edge.

```
Figure 3.7: Final result of multiscale snake, whereby the final contour is drawn on the gradient magnitude image.
```

### 3.4 Gradient Vector Flow

An alternative snake method is the gradient vector flow snake that was developed by Chenyang Xu and Jerry L. Prince [30]. It handles the small attraction range of the snake as well. The gradient vector field is used as another force derived from the image that works on the snake, but as opposed to the previous force derived from the image, namely the gradient magnitude, it is not a scalar, but a vector. By applying the gradient vector field the snake not only has a larger attraction range, but enters concavities in the border more easily.

The gradient of an image can be depicted as a vector field in which the derivative in the x direction and the derivative in the y direction determine the direction and size of the vector. The gradient vector flow field can now be visualized as the gradient of an image diffused out over the image. Near the edge the gradient vector flow field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.

The gradient vector flow field is defined as the vector field $\vec{v}(x, y) = (u(x, y), v(x, y))$ that minimizes the following energy function.

$$\varepsilon = \int \int_\Omega \left( \mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) + \left\| \nabla f - \nabla \vec{v} \right\|^2 \right) \, dx \, dy \quad (3.7)$$

In which $\mu$ is a constant, $u_x$, $u_y$, $v_x$, and $v_y$, are the derivatives of $u$ and $v$ in the x and y direction respectively and $f$ is the image gradient magnitude.

When $\nabla f$ is small, which occurs in homogeneous areas, the energy is a result of the partial derivatives of the vector field. When the energy is minimized the derivatives of the vector field become smaller, thus the vector field points towards the edge, as in homogeneous areas the field varies smoothly towards the edge.
field itself becomes smoother. When $\nabla^2 f$ is big the vector field that causes the energy to minimize approaches $\nabla f$. The parameter $\mu$ determines the trade-off between the first term and the second term on the right side of equation 3.4, thus effectively how much smoothing there is with respect to the pointing towards the edges.

This can be implemented by solving the following equations.

$$\mu \nabla^2 u - (u - f_x) (f_x^2 + f_y^2) = 0$$  \hspace{1cm} (3.8)

$$\mu \nabla^2 v - (v - f_y) (f_x^2 + f_y^2) = 0$$  \hspace{1cm} (3.9)

In which $\nabla^2 i$ is the Laplacian operator, $f_x$ and $f_y$ are the derivatives in the x and y direction of the image respectively.

Now u and v can be treated as functions dependent on time. By incorporating the following numerical approximations the solution can be determined for the discrete case.

$$u_i = \frac{1}{\Delta t} (u_{i,j}^{n+1} - u_{i,j}^n) + \frac{1}{\Delta x \Delta y} (u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4 u_{i,j}) \nabla^2 u =$$

$$= \frac{1}{\Delta x \Delta y} (v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4 v_{i,j})$$

In which $\Delta x$ and $\Delta y$ are the distances between the pixels in the x and y direction respectively and $\Delta t$ is the timestep.

The resulting iterative solution for the gradient vector flow field now becomes:

$$u_{i,j}^{n+1} = (1 - (f_x^2 + f_y^2)) u_{i,j}^n +$$

$$r(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1} - 4 u_{i,j}) \nabla^2 u +$$

$$v_{i,j}^{n+1} = (1 - (f_x^2 + f_y^2)) u_{i,j}^n +$$

$$r(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1} - 4 v_{i,j}) \nabla^2 v$$

In which $r = \frac{\mu \Delta t}{\Delta x \Delta y}$.

To guarantee the convergence of the gradient vector flow the timestep is under the following restriction.

$$\Delta t \leq \frac{\Delta x \Delta y}{4 \mu}$$  \hspace{1cm} (3.12)

When $\Delta x$ and $\Delta y$ are bigger, thus the image is coarser bigger timesteps can be taken, but when $\mu$ is large and it is desired that the gradient vector flow field is very smooth, small timesteps are required.

The gradient vector flow field has been determined for the test image created before.

```plaintext
fx = gD[imgrad, 1, 0, 5]; fy = gD[imgrad, 0, 1, 5]; b = fx^2 + fy^2;
c1 = b fx; c2 = b fy; dx = 1; dy = 1; \mu = 1; \Delta t = \frac{\Delta x \Delta y}{4 \mu}; x = \frac{\Delta t \mu}{\Delta x \Delta y};
teller = 1; u = Table[0, {x}, {y}]; v = Table[0, {x}, {y}];

While[teller <= 5000, u = (1 - b \Delta t) u +
    r (RotateLeft[u] + RotateLeft /@ u + RotateRight[u] + RotateRight /@ u - 4 u) +
    c1 \Delta t;
    v = (1 - b \Delta t) v + r (RotateLeft[v] + RotateLeft /@ v +
        RotateRight[v] + RotateRight /@ v - 4 v) + c2 \Delta t;
    teller++];
```
To display the vector field in an understandable way the image is downsized by taking one out of 4 pixels.

```
downSize[im_, n_] := Module[{dim},
   If[n ≤ 0, n = 1];
   dim = Dimensions[im];
   im[[Round[Range[1, dim[[1]]], n]], Round[Range[1, dim[[2]], n]]]]
udown = downSize[u, 4]; vdown = downSize[v, 4];
gvff = ListPlotVectorField[
   Transpose[{Reverse[udown], Reverse[vdown]}, {3, 1, 2}], ScaleFactor → 1];
DisplayTogetherArray[{{gvff0, gvff100}, {gvff1000, gvff5000}}];
```

Figure 3.8: Gradient vector flow field after 0, 100, 1000, 5000 iterations.

The gradient vector flow field is now incorporated into the snake by adding it to the increment the contour point has to make. Thus it is considered an extra force. The code for this is given in the appendix. A simple example is shown in figure 3.9.

Figure 3.9: A simple result of the gradient vector flow snake.
3.5 Balloon Snake

Pressured snakes have been proposed by Cohen [5] to increase the attraction range by adding a pressure force. When the model deforms, the pressure force keeps inflating or deflating the snake until it is stopped by the external energy. It removes the requirement to initialize the model near or outside the desired object boundaries. Deformable models that use pressure forces are also known as balloons. The pressure force is defined as:

\[ F_p(X) = \kappa \mathbf{N}(X) \]  \hspace{1cm} (3.13)

where \( \mathbf{N} \) is the inward unit normal of the model at the point \( X \) and \( \kappa \) is a weighting constant parameter. The sign of \( \kappa \) determines whether to inflate or to deflate the snake. Its value gives the strength of the pressure force. This force helps the snake to trespass isolated weak image edges, and counters its tendency to shrink. The resulting snake is more robust to the initial position and image noise, but human intervention is needed to decide whether an inflationary or deflationary force is needed. Furthermore, one problem with the pressure snake is that medical images usually contain weak, broad edges, that render snake models ineffective. Unfortunately, an image must have strong edges to overcome the pressure force and allow the model to reach equilibrium and stop. Consequently, it is not always possible to extract features using pressure force. Statistical snakes [12] overlay this limitation and prevent human intervention. Region information is now used to define the sign of \( \kappa \), depending on whether the model is inside or outside the desired object.

The balloon force of the statistical snake used is this project is the binary pressure force defined by Ivins and Porrill [12]. A seed region is selected within the area to be segmented, and the mean \( \mu \) and standard deviation \( \sigma \) of intensity in this area are computed. The pressure force makes the snake expand or contract itself according to the statistical values: if \( |I(x,y)-\mu| \leq k\sigma \) then the snake will be inflated, else it will contract (k is a constant). This means that the statistical snake will expand until it encounters pixels that are outside the area to be segmented, relative to a seed region. When that happens, the pressure force is reversed to make the model contract.

A drawback of using pressure forces is that sometimes the curve crosses itself and forms loops. That is why it is necessary to merge points if they are too close, and to keep the distance between consecutive points nearly constant. In the appendix an implementation is given.

![Figure 3.10: Result of balloon snake on a simple test image with initial contour (left), result (middle) and result on gradient magnitude image (right).](image)
3.6 A Semi-Automatic Initialization

Now a semi-automatic way to define the first contour is created. This method consists in selecting the center of the area to be segmented, and a rectangular zone containing this area. This is why the bottom left and the top right corners of this box must also be selected. Then the pixel with the maximum gradient magnitude value is determined in a number of directions uniformly distributed around the centre, but inside the box.

Figure 3.11: Seek on each ray the pixel of maximum intensity.

3.7 Applying the snake algorithm and discussion

In this section the basic snake, the multiscale snake and the gradient vector flow snake are applied on several test and real images to test the robustness of the algorithm. The results are compared and discussed.

First it is tested on an artificial image of a cross with noise added as was also done in chapter two. In figure 3.12 the result of adding 100% noise to the test image is given for the three snake methods. In reality it is very unlikely to get images with 100% noise added, therefor if the snake algorithm performs well on this it is a good sign. The snake parameters are chosen as follows: $\alpha = 1$ for the continuity, $\beta = 0.5$ for the curvature, $\gamma = 1$ for the external force. The multiscale snake uses the initial $\sigma$ of 20 and decreases with 2. The gradient vector flow snake adds the gradient vector field force with a parameter $\delta = 0.1$.

DisplayTogetherArray[{basic, mscale, gvf}];

Figure 3.12: Result of the various snakes on an image with 100% noise added.

As can be seen the basic snake performs badly. The other two snake methods segment the cross reasonably well.

The noise causes local maxima in the gradient magnitude image and in case of the basic snake the contour points stop in a configuration that may not be on the desired edge, but on an edge caused by the noise. In case
of the multiscale snake the noise is blurred as well. This means that the multiscale snake first adapts to the larger structures in the image and this causes the contour points to converge closely to the edge. When the scale is reduced the details of the object are close by the contour points, which reduces the chance that the contour points of the multiscale snake get stuck in an edge caused by the noise. In case of the gradient vector flow snake the gradient of the image is smoothed out, in which strong edges are kept strong. This smoothing has a similar effect as blurring. In case of even more and stronger noise the gradient vector flow might perform less than the multiscale snake as the gradient caused by the noise will no longer be smoothed away by the gradient of the desired edge. In the multiscale approach however the noise will be evened out by blurring, so that only the larger structures in the image remain, as shown in figure 3.13.

\[
\text{DisplayTogetherArray}[[\text{mscale, gvf}]]; \\
\]

Figure 3.13: Result of multiscale (left) and gvf (right) on 300 % noise.

Next the various snake algorithms are tested on an image with a gap in the gradient magnitude image. The result of the basic snake, the multiscale snake and the gradient vector flow snake on an image with a gap in the gradient magnitude image of 2 pixels is given in figure 3.14.

\[
\text{DisplayTogetherArray}[[\text{basic2, mscale2, gvf2}]] \\
\]

Figure 3.14: Result of various snakes on an image with a gap in the gradient magnitude image.

As can be seen in figure 3.14 all the different snakes handle a gap in the gradient magnitude image very well. The gap is closed by the contour as if there is no gap at all.

This is exactly one of the reasons snake algorithms were developed, therefore it is no surprise that all the snake methods perform well. The internal energies ensure the spanning of gaps in the gradient magnitude image as seen in figure 3.14. The continuity energy makes sure that the contour points are equidistant on the contour, preventing contour points moving too far from other contour points, and the curvature energy keeps the contour rigid, making sure that a contour point stays more or less in line with the contour points before and after it. There are however limits to the size of gaps that can be closed by the snake. When the gaps are larger than several contour points the danger arises that these contour points move inward too far, before they are stopped by the internal energies.
Now let us test the performance of the snake on a real image that has a gradient added to it, as explained in chapter two. In figure 3.15 the result of the basic snake, the multiscale snake and the gradient vector flow snake on the right ventricle of a heart of a mouse is given. The initial contour is drawn close to the actual edge to ensure the right edge is detected.

\[
\text{DisplayTogetherArray}\{\text{basic, mscale, gvf}\}
\]

Figure 3.15: Result of various snakes on a real image with a gradient added.

As can be seen in figure 3.15 the snake performs well even when a gradient is added to the image. This can be explained in the same way as was done for the watershed. The snake performs on the gradient magnitude image. As such the added gradient is filtered out when the gradient magnitude is calculated. There might be small irregularities on small scale, when the added gradient decreases an edge, but this will only be on already weak edges.

As for the performance on a real image the snake performs reasonably well. A few things can be noticed by comparing the different snake methods. The snakes do not enter the concavity in the right ventricle, indicated by the blue arrow, very well, but the gradient vector flow has less problems than the other two methods and enters the concavity slightly more. But the result is still not completely on the edge of the concavity. The snakes do not completely enter the concavity as a result of the internal energies. Thus there is a trade-off between entering concavities and closing gaps. If the internal energies are very small the concavity will be detected better, but the snake will be more susceptible to entering gaps in stead of spanning them.

In both the basic snake as in the gradient vector flow one contour point has stopped on the wrong edge. In the multiscale snake this is not the case. This is because in the multiscale snake the contour points are first guided by the strongest edges in the neighborhood. As the edge of the right ventricle is stronger than the edge of the heart wall, because of the contrast fluid in the ventricles, the contour points are already moved near the edge that is desired. They are adjusted slightly during the decrease of the scale to fit the edge more precisely. This might seem a good thing, but can also cause problems. When an object with weak edges must be detected in the presence of objects with strong edges, blurring might cause the contour points to be located on the strong edges first. Then when the scale is decreased, the contour points will not be able to converge on the weak edge. Strong edges or points near an object of interest only effect the snake if one of the contour points come within its neighborhood. Because the contour point evolution is only dependent on the direct neighborhood of the contour point.
Another interesting test is to determine what happens when the snake is initialized around more than just one object. This has been tested by creating a test image with two white circles on a black background and initializing the contour points around both the circles. The result is shown in figure 3.16.

\[
\text{DisplayTogetherArray\{\{\text{basic, mscale, gvf}\}\}}
\]

![Image](image.png)

Figure 3.16: Result of various snakes on two objects in the image.

As can be seen in figure 3.16 the snake method cannot determine that there is more than one object. The multiscale and the gradient vector flow snake bring the two contour lines, that gap the distance between the two circles, closer to each other than the basic contour, but are unable to remove them. This is a serious limitation of the snake algorithm. The method as given here cannot merge two snake contours when the touch each other or separate one contour into two or more separate contours. Special bookkeeping is required if this feature is desired, because the contour points that belong to one contour must be stored independently from another contour and when they merge it must be determined which contour point follows which. As the method is now it can only be used to segment one object in an image.

### 3.8 Discussion And Future Enhancements

Overall the snake method works well to segment one object in an image. It performs well on images with noise, objects with gaps in the edges and on images with a gradient added to the image. The speed of the implementation in Mathematica is also very well, the snake takes less than a minute to converge. It is however not an automatic segmentation. The initial contour points must be placed manually and they must be placed reasonably close to the edge of the object that is to be segmented. Furthermore it does not follow the edge very well if the edge has a sudden concave shape. These concavities are followed a little better with the gradient vector flow snake, but this also has limitations.

One limitation of the snake is that it cannot merge two contours to one or split one contour to two contours. This can be helped by storing the contour points in different snakes. Whenever contours intersect this should be detected and how the contour points are stored should be changed in such a way that the contour points of separate snakes are stored separately. The ordering of the contour points should be altered to ensure that a contour point that comes after another contour point is also stored as such. It can be seen that this involves meticulous bookkeeping. Another way of handling merging and splitting of the contour is given in chapter four.
Another addition to the snake algorithm can be to make the number of contour points increase or decrease according to the demand. Whenever the contour points start to lie far apart contour points can be added in between the existing contour points. Whenever the contour points start to lie very close to each other contour points can be removed. This ensures that when a contour that is drawn outward to the boundaries keeps a good representation of this boundary and does not cut corners.

The snake can also be implemented for three dimensional images. This can be done in several ways. First is by simply taking two dimensional slices from the three dimensional image and segment the object that is desired with two dimensional snakes. Then build up the three dimensional form from the two dimensional contours. The initialization of the first contour can be done by hand, but the next initialization can be taken to be the end result of the previous slice.

Another method is done by letting the contour points move in three dimensions directly. This again requires a lot of bookkeeping to determine the neighbors and the second neighbors, which are required to calculate the continuity and curvature energies. The result will be a surface and can immediately be represented in three dimensions.